



\*\*FILE\*\*ID\*\*RM3DELETE

K 10

RRRRRRRR RRRRRRRR MM MM 333333 DDDDDDDD EEEEEEEEEE LL EEEEEEEEEE TTTTTTTTTT EEEEEEEEEE  
RR RR MMMM MMMM 33 33 DD DD EE EE LL EE TT EE  
RR RR MMMM MMMM 33 33 DD DD EE EE LL EE TT EE  
RR RR MM MM MM 33 33 DD DD EE EE LL EE TT EE  
RR RR MM MM 33 33 DD DD EEEEEEEEEE LL EEEEEEEEEE  
RRRRRRRR MM MM 33 33 DD DD EEEEEEEEEE LL EEEEEEEEEE  
RRRRRRRR MM MM 33 33 DD DD EEEEEEEEEE LL EEEEEEEEEE  
RR RR MM MM 33 33 DD DD EE EE LL EE TT EE  
RR RR MM MM 33 33 DD DD EE EE LL EE TT EE  
RR RR MM MM 33 33 DD DD EE EE LL EE TT EE  
RR RR MM MM 333333 DDDDDDDD EEEEEEEEEE LLLLLLLLLL EEEEEEEEEE TT EEEEEEEEEE  
RR RR MM MM 333333 DDDDDDDD EEEEEEEEEE LLLLLLLLLL EEEEEEEEEE TT EEEEEEEEEE

LL IIIII SSSSSSS  
LL IIIII SSSSSSS  
LL II SS  
LL II SS  
LL II SS  
LL II SSSSS  
LL II SSSSS  
LL II SS  
LL II SS  
LL II SS  
LLLLLLLLL IIIII SSSSSSS  
LLLLLLLLL IIIII SSSSSSS

RM  
VO

```
1 0001 0 MODULE RM3DELETE (LANGUAGE (BLISS32) .
2 0002 0 IDENT = 'V04-000'
3 0003 0 )
4 0004 1 BEGIN
5 0005 1 ****
6 0006 1 *
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *
27 0027 1 ****
28 0028 1 ++
29 0029 1 ++
30 0030 1
31 0031 1 FACILITY: RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1 This module handles the deletion of index sequential records.
35 0035 1
36 0036 1
37 0037 1
38 0038 1 ENVIRONMENT:
39 0039 1
40 0040 1 VAX/VMS OPERATING SYSTEM
41 0041 1
42 0042 1 --
43 0043 1
44 0044 1
45 0045 1 AUTHOR: Todd M. Katz CREATION DATE: 14-Jul-1982
46 0046 1
47 0047 1
48 0048 1 MODIFIED BY:
49 0049 1
50 0050 1 V03-025 JWT0181 Jim Teague 15-May-1984
51 0051 1 RM$SQUISH moves too many bytes when squishing the
52 0052 1 the data portion out of deleted records.
53 0053 1
54 0054 1 V03-024 DAS0001 David Solomon 25-Mar-1984
55 0055 1 Fix broken branch to RMSRU_JOURNAL3.
56 0056 1
57 0057 1 V03-023 MCN0003 Maria del C. Nasr 04-Apr-1983
```

58      0058 1      Change linkage of RM\$NULLKEY to RL\$JSB.  
59      0059 1  
60      0060 1  
61      0061 1  
62      0062 1  
63      0063 1  
64      0064 1  
65      0065 1  
66      0066 1  
67      0067 1  
68      0068 1  
69      0069 1  
70      0070 1  
71      0071 1  
72      0072 1  
73      0073 1  
74      0074 1  
75      0075 1  
76      0076 1  
77      0077 1  
78      0078 1  
79      0079 1  
80      0080 1  
81      0081 1  
82      0082 1  
83      0083 1  
84      0084 1  
85      0085 1  
86      0086 1  
87      0087 1  
88      0088 1  
89      0089 1  
90      0090 1  
91      0091 1  
92      0092 1  
93      0093 1  
94      0094 1  
95      0095 1  
96      0096 1  
97      0097 1  
98      0098 1  
99      0099 1  
100     0100 1  
101     0101 1  
102     0102 1  
103     0103 1  
104     0104 1  
105     0105 1  
106     0106 1  
107     0107 1  
108     0108 1  
109     0109 1  
110     0110 1  
111     0111 1  
112     0112 1  
113     0113 1  
114     0114 1

V03-022 TMK0013 Todd M. Katz 26-Mar-1983  
Change the linkage for RMSRU\_JOURNAL3 from RL\$RABREG\_467 to  
RL\$RABREG\_67.

V03-021 MCN0002 Maria del C. Nasr 24-Mar-1983  
More linkages reorganization.

V03-020 RAS0135 Ron Schaefer 17-Mar-1983  
Fix spelling of RJRS\_DELETE -> RJRS\_DELETE.

V03-019 TMK0012 Todd M. Katz 16-Mar-1983  
Change the linkage for RMSRU\_JOURNAL3 from RL\$RABREG\_67 to  
RL\$RABREG\_467.

V03-018 TMK0011 Todd M. Katz 16-Mar-1983  
Change the symbol RMSR\$\_DELETE to RJRS\_DELETE.

V03-017 MCN0001 Maria del C. Nasr 24-Feb-1983  
Reorganize linkages

V03-016 TMK0010 Todd M. Katz 08-Jan-1983  
Add support for Recovery Unit Journalling and RU ROLLBACK  
Recovery of ISAM files. This support includes:

1. The restructuring of RM\$DELETE3B so that the primary data record is unpacked and available for RU journalling before any part of the file is permanently modified.
2. The RU Journalling of all \$DELETEs which occur on RU Journalled files within Recovery Units.
3. Modifications to RMSDELETE\_RRV, RMSSQUISH\_SIDR, and RMSDELETE\_UDR so that no space is reclaimed when records of RU journalled files are \$DELETEd within Recovery Units. The RRV, primary data record, or SIDR array element is just marked RU\_DELETE instead.
4. Modifications to RM\$DELETE\_RRV, RMSSQUISH\_SIDR, and RMSDELETE\_UDR so that RRVs, primary data records and SIDR array elements maybe un-deleted during ROLLBACK of prematurely terminated or aborted Recovery Units.
5. The addition of a second parameter (SCAN) to RMSSQUISH\_SIDR. If this parameter is 1 on entry, RMS will scan the entire SIDR array looking for non-deleted elements even if no duplicates are allowed in the key of reference. If SCAN is 0 RMS will immediately delete the entire SIDR as was the case previously.

V03-015 TMK0009 Todd M. Katz 05-Jan-1983  
The routine RMSDELETE\_SIDR no longer calls the routine RMSFND\_SDR\_ARRAY to position to the SIDR element it is to delete. It now performs its own positioning.

V03-014 TMK0008 Todd M. Katz 07-Dec-1982

115 0115 1  
116 0116 1  
117 0117 1  
118 0118 1  
119 0119 1  
120 0120 1  
121 0121 1  
122 0122 1  
123 0123 1  
124 0124 1  
125 0125 1  
126 0126 1  
127 0127 1  
128 0128 1  
129 0129 1  
130 0130 1  
131 0131 1  
132 0132 1  
133 0133 1  
134 0134 1  
135 0135 1  
136 0136 1  
137 0137 1  
138 0138 1  
139 0139 1  
140 0140 1  
141 0141 1  
142 0142 1  
143 0143 1  
144 0144 1  
145 0145 1  
146 0146 1  
147 0147 1  
148 0148 1  
149 0149 1  
150 0150 1  
151 0151 1  
152 0152 1  
153 0153 1  
154 0154 1  
155 0155 1  
156 0156 1  
157 0157 1  
158 0158 1  
159 0159 1  
160 0160 1  
161 0161 1  
162 0162 1  
163 0163 1  
164 0164 1  
165 0165 1  
166 0166 1  
167 0167 1  
168 0168 1  
169 0169 1  
170 0170 1  
171 0171 1

Change the order in which the various parts of a record are deleted during a \$DELETE. First, eliminate the RRV. Next eliminate the user data record. Finally, the alternate keys which are represented in the primary data record are removed. Previously, the SIDRs were eliminated before the primary data record, and during this time a lock was kept on the primary data bucket. This meant that a bucket lock was being held for quite a long time, and that the routine that positioned to a primary data record by means of an alternate index had to be enhanced with a very complex and very large SIDR re-positioning routine, so that the 1.5 SIDR deadlock case would not exist in version 4. Changing the order of events that take place during a \$DELETE allowed a change in the bucket lock strategy which had the dual benefits of eliminating the 1.5 SIDR deadlock case without the expensive SIDR re-positioning code, and reducing the amount of time a lock on the primary data bucket is kept to a minimum - which is an overall ISAM design goal.

This change is not without its cost. The reason why the old strategy was originally implemented, was so that the primary data record would be available for the extraction of the alternate keys so that the corresponding SIDRs could be eliminated. Changing the bucket locking strategy such that the primary data record is deleted and the bucket is released before the SIDRs are deleted means that the primary data record must be saved in an auxiliary record buffer before it is deleted so that it will be available for alternate key extraction. However, this change is not as expensive as it might seem because if the file's prologue version is 3, the primary data record would have to be unpacked into this same record buffer before the keys could be extracted anyway. Thus, it was a simple matter of unpacking either sooner or later. Any additional cost incurred by this new strategy is born solely by prologue 1 and 2 files which previously could extract the alternate keys without moving the primary data record, and now must perform an additional MOVC3. However, the benefits derived from this new strategy more than outweigh the cost of this additional MOVC3 required in the case of a prologue version which will hopefully fade out of use.

V03-013 TMK0007 Todd M. Katz 06-Dec-1982  
The routine RMSSQUISH\_SIDR was recovering the space occupied by a SIDR whenever duplicates were allowed and all the elements in the SIDR were deleted even if the SIDR occupied the physically last position in the SIDR bucket. This had the possibility of creating totally empty SIDR buckets, and the encountering of a totally empty SIDR bucket during a positioning for insertion when duplicates are allowed can not always be correctly handled. Thus, a bug existed in the \$DELETE code which had capability of corrupting SIDR indicies.

To fix this code I have decided that the space occupied by the physically last SIDR in the bucket can never be recovered even if all the elements in the array are deleted when duplicates alternate keys are allowed. At best, if the file is a prologue 3 file, and the element is not the first element in the SIDR array, the space occupied by the RRV pointer can be recovered.

172 0172 1  
173 0173 1  
174 0174 1  
175 0175 1  
176 0176 1  
177 0177 1  
178 0178 1  
179 0179 1  
180 0180 1  
181 0181 1  
182 0182 1  
183 0183 1  
184 0184 1  
185 0185 1  
186 0186 1  
187 0187 1  
188 0188 1  
189 0189 1  
190 0190 1  
191 0191 1  
192 0192 1  
193 0193 1  
194 0194 1  
195 0195 1  
196 0196 1  
197 0197 1  
198 0198 1  
199 0199 1  
200 0200 1  
201 0201 1  
202 0202 1  
203 0203 1  
204 0204 1  
205 0205 1  
206 0206 1  
207 0207 1  
208 0208 1  
209 0209 1  
210 0210 1  
211 0211 1  
212 0212 1  
213 0213 1  
214 0214 1  
215 0215 1  
216 0216 1  
217 0217 1  
218 0218 1  
219 0219 1  
220 0220 1  
221 0221 1  
222 0222 1  
223 0223 1  
224 0224 1  
225 0225 1  
226 0226 1  
227 0227 1  
228 0228 1

This fix which I have implemented by re-writing the routine RM\$SQUISH\_SIDR (both to implement the fix and to optimize the existing code) guarantees both that empty SIDR buckets can never be created when duplicate SIDRs are allowed, and that NRP positioning context is maintained.

V03-012 TMK0006 Todd M. Katz 14-Nov-1982  
The routine RM\$DELETE\_UDR no longer has to return a value. Previously, it was returning a value because the routine that was responsible for reclaiming space occupied by records that were just marked deleted needed to know whether or no an RRV had been created in the place of the reclaimed record. This is no longer the case, as that routine has been modified to no longer require this piece of information.

V03-011 TMK0005 Todd M. Katz 12-Nov-1982  
The routine RM\$FND\_SDR\_ARRAY requires as implicit input the key size of the SIDR if it is to position to in IR\$SB KEYSZ. The routine RM\$DELETE\_SIDR was not setting up the IRAB cell with the key size before calling this routine. Therefore, the possibility existed that RM\$FND\_SDR\_ARRAY would position to the wrong SIDR array, which would then be deleted. This in fact has been seen, during the course of an \$UPDATE when the old SIDRs that have been changed are removed, and this fix corrects this problem.

V03-010 TMK0004 Todd M. Katz 11-Nov-1982  
When SIDRs must be deleted and the file is a prologue 3 file, the record must be unpacked so that the alternate keys can be extracted. If RMS positioned by the primary key of reference then it will already have a fully expanded copy of the primary key in keybuffer 1, and it can use this in the unpacking of the record instead of scanning the bucket to re-expand the primary key when primary key compression is enabled. There is one case when it can not use the primary key in keybuffer 1 like this, and that is when the record being deleted is not the same as the current primary data record. This happens when RMS randomly \$FINDs a record since this operation does not update the NRP context. I was not checking for this case and this fix remedies this.

V03-009 TMK0003 Todd M. Katz 06-Oct-1982  
When I completely re-wrote this routine (TMK0001), I broke the deletion of prologue 3 fixed length records, in certain cases, because I had assumed that all prologue 3 records included as part of their record overhead a record size field that needs to be updated when the portion of the prologue 3 primary data record occupied by the data is reclaimed. I thought I had fixed this in TMK0002 (although I forgot to mention it in the audit trial), but actually all I did was fix one \$DELETE case and break others that occur more frequently. What I did was to make the assumption that all fixed length prologue 3 records do not include a record size field. This too is incorrect. Actually, if a prologue 3 record with fixed length records has either key or data compression (or both) enabled, then there is a record size field present as part of the record overhead. If both compression types are disabled and the

: 229 0229 1 record is fixed size then there is no need for a record size  
230 0230 1 field and one is not present. I was not checking any compression  
231 0231 1 bits, but rather, just for a fixed length record format, before  
232 0232 1 deciding whether or there was a record size field to update and  
233 0233 1 this is what caused the problem in TMK0002.  
234 0234 1  
235 0235 1 V03-008 TMK0002 Todd M. Katz 04-Sep-1982  
236 0236 1 Add support for prologue 3 SIDRs. This involves changes  
237 0237 1 only to the routine RM\$SQUISH\_SIDR.  
238 0238 1  
239 0239 1 The field IRB\$B\_SRCHFLAGS is now a word. Change all references  
240 0240 1 to it.  
241 0241 1  
242 0242 1 V03-007 KBT0162 Keith B. Thompson 21-Aug-1982  
243 0243 1 Reorganize psects  
244 0244 1  
245 0245 1 V03-006 TMK0001 Todd M. Katz 02-Jul-1982  
246 0246 1  
247 0247 1 New version of \$DELETE. This module now incorporates all  
248 0248 1 the routines which were formerly in RM3DELSDR.  
249 0249 1  
250 0250 1 \*\*\*\*\*  
251 0251 1  
252 0252 1 LIBRARY 'RMSLIB:RMS';  
253 0253 1  
254 0254 1 REQUIRE 'RMSSRC:RMSIDXDEF';  
255 0319 1  
256 0320 1 ! Define default PSECTS for code.  
257 0321 1  
258 0322 1 PSECT  
259 0323 1 CODE = RM\$RMS3(PSECT\_ATTR),  
260 0324 1 PLIT = RM\$RMS3(PSECT\_ATTR);  
261 0325 1  
262 0326 1 ! Linkages.  
263 0327 1  
264 0328 1 LINKAGE  
265 0329 1 L\_ERROR\_LINK1,  
266 0330 1 L\_JSB,  
267 0331 1 L\_JSB01,  
268 0332 1 L\_LINK 7 10 11,  
269 0333 1 L\_PRESERVE1,  
270 0334 1 L\_RABREG,  
271 0335 1 L\_RABREG\_4567,  
272 0336 1 L\_RABREG\_567,  
273 0337 1 L\_RABREG\_67,  
274 0338 1 L\_RABREG\_7,  
275 0339 1 L\_REC\_OVRL,  
276 0340 1 L\_SIDR\_FIRST,  
277 0341 1  
278 0342 1 ! Local Linkage  
279 0343 1  
280 0344 1 RL\$DEL\_ALL\_SIDR = JSB ()  
281 0345 1 : GLOBAL (R\_REC\_ADDR,R\_IDX\_DFN,COMMON\_RABREG),  
282 0346 1 RL\$SQUISH\_DATA = JSB ()  
283 0347 1 : GLOBAL(R\_REC\_ADDR,R\_BKT\_ADDR,R\_IDX\_DFN,R\_IFAB);  
284 0348 1  
285 0349 1 ! External Routines

```
286 0350 1 !  
287 0351 1 EXTERNAL ROUTINE  
288 0352 1 RMSCLEAN BDB  
289 0353 1 RMSCSEARCH TREE  
290 0354 1 RMSEXPNAD REYD  
291 0355 1 RMSEXT ARRY RFA  
292 0356 1 RMSFIND BY ID  
293 0357 1 RMSGET NEXT KEY  
294 0358 1 RMSGETNXT ARRAY  
295 0359 1 RMSKEY DESC  
296 0360 1 RMSMOVE  
297 0361 1 RMSNULLKEY  
298 0362 1 RMSREC OVHD  
299 0363 1 RMSRECORD_ID  
300 0364 1 RMSRECORD_KEY  
301 0365 1 RMSRECORD_VBN  
302 0366 1 RMSRLSBKT  
303 0367 1 RMSRU JOURNAL3  
304 0368 1 RMSSIDR_END  
305 0369 1 RMSSIDR_FIRST  
306 0370 1 RMSUNPACK REC  
307 0371 1 RMSUPDDELCOM  
308 0372 1 ! Forward Routines  
309 0373 1 FORWARD ROUTINE  
310 0374 1  
311 0375 1 FORWARD ROUTINE  
312 0376 1 RMSDELETE_RRV  
313 0377 1 RMSDELETE_SIDR  
314 0378 1 RMSDELETE_UDR  
315 0379 1 RMSSQUISH_DATA  
316 0380 1 RMSSQUISH_SIDR  
:  
: RL$ERROR LINK1,  
: RL$RABREG_67,  
: RL$JSB01,  
: RL$RABREG_67,  
: RL$RABREG_567,  
: RL$LINK 7-10_11,  
: RL$RABREG_67,  
: RL$RABREG_7,  
: RL$PRESERVE1,  
: RL$JSB,  
: RL$REC OVHD,  
: RL$RABREG_67,  
: RL$PRESERVE1,  
: RL$PRESERVE1,  
: RL$PRESERVE1,  
: RL$PRESERVE1,  
: RL$RABREG_67 ADDRESSING_MODE( LONG_RELATIVE ),  
: RL$RABREG_67,  
: RL$SIDR FIRST,  
: RL$JSBOT,  
: RL$RABREG_67;
```

```
318 0381 1 %SBTTL 'RMSDEL_ALL_SIDR'  
319 0382 1 ROUTINE RMSDEL_ALL_SIDR (RECORD_SIZE) : RLSDEL_ALL_SIDR NOVALUE =  
320 0383 1  
321 0384 1 ++  
322 0385 1  
323 0386 1 FUNCTIONAL DESCRIPTION:  
324 0387 1  
325 0388 1 The purpose of this routine is to delete every SIDR array element  
326 0389 1 pointing to the the current primary data record. Towards this goal  
327 0390 1 every secondary key represented in the current primary data record  
328 0391 1 is in turn extracted from the current primary data record which has  
329 0392 1 been saved (in an unpacked form if prologue 3) in a record buffer, used  
330 0393 1 to position to the SIDR array element pointing to the current primary  
331 0394 1 data record in the appropriate index, and that array element is  
332 0395 1 deleted. If the current primary data record does not possess one or  
333 0396 1 more secondary keys either because the record is not of sufficient size  
334 0397 1 or the key is null, or if a fast delete is requested and duplicates of  
335 0398 1 one or more secondary keys are allowed, then the deletion of those  
336 0399 1 secondary keys are bypassed.  
337 0400 1  
338 0401 1 CALLING SEQUENCE:  
339 0402 1  
340 0403 1 RMSDEL_ALL_SIDR()  
341 0404 1  
342 0405 1 INPUT PARAMETERS:  
343 0406 1  
344 0407 1 RECORD_SIZE - size of the user data record in IRB$L_RECBUF  
345 0408 1  
346 0409 1 IMPLICIT INPUTS:  
347 0410 1  
348 0411 1  
349 0412 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1  
374 0437 1  
350 0413 1  
351 0414 1  
352 0415 1  
353 0416 1  
354 0417 1  
355 0418 1  
356 0419 1  
357 0420 1  
358 0421 1  
359 0422 1  
360 0423 1  
361 0424 1  
362 0425 1  
363 0426 1  
364 0427 1  
365 0428 1  
366 0429 1  
367 0430 1  
368 0431 1  
369 0432 1  
370 0433 1  
371 0434 1  
372 0435 1  
373 0436 1<br
```

```
375 0438 1 !--  
376 0439 1  
377 0440 2 BEGIN  
378 0441 2  
379 0442 2 BUILTIN  
380 0443 2 AP;  
381 0444 2  
382 0445 2 EXTERNAL REGISTER  
383 0446 2 COMMON_RAB_STR,  
384 0447 2 R_IDX_DFN_STR,  
385 0448 2 R_REC_ADDR_STR;  
386 0449 2  
387 0450 2 LABEL  
388 0451 2 BLOCK;  
389 0452 2  
390 0453 2 ! Delete all of the secondary keys present in the current user data record.  
391 0454 2 !  
392 0455 2 WHILE RMSGET_NEXT_KEY()  
393 0456 2 DO  
394 0457 2  
395 0458 2 ! Each secondary key in the file will in turn become the "current"  
396 0459 2 secondary key for the purpose of deleting its representative in the  
397 0460 2 current primary data record from the appropriate index.  
398 0461 2  
399 0462 3 BLOCK: BEGIN  
400 0463 3  
401 0464 3 ! If a fast-delete is requested, terminate the deletion of the current  
402 0465 3 secondary key only if this secondary key allows duplicates. If this  
403 0466 3 secondary key does not allow duplicates, then a fast delete of it can  
404 0467 3 not be done, since the error caused by a later attempt to insert a  
405 0468 3 record with a secondary key that is a duplicate of this one would go  
406 0469 3 undetected.  
407 0470 3  
408 0471 3 IF .RAB[RABSV_FDL]  
409 0472 3 AND  
410 0473 3 .IDX_DFN[IDX$V_DUPKEYS]  
411 0474 3 THEN  
412 0475 3 LEAVE BLOCK;  
413 0476 3  
414 0477 3 ! Check that the current primary data record is of a sufficient size to  
415 0478 3 include the current secondary key. If it is not, terminate the  
416 0479 3 deletion process for this secondary key.  
417 0480 3  
418 0481 3 IF .RECORD_SIZE<0, 16> LSSU .IDX_DFN[IDX$W_MINRECSZ]  
419 0482 3 THEN  
420 0483 3 LEAVE BLOCK;  
421 0484 3  
422 0485 3 ! In preparation for positioning to the SIDR array element for this  
423 0486 3 secondary key of the current primary data record, the secondary key  
424 0487 3 must be extracted into keybuffer 2.  
425 0488 3  
426 0489 3 REC_ADDR = .IRAB[IRB$L_RECBUF];  
427 0490 3  
428 0491 3 ! If this secondary key for the current primary data record is null,  
429 0492 3 there will not be a SIDR array element in this index pointing to the  
430 0493 3 current primary data record. Therefore, there is no need to continue  
431 0494 3 with the process of deleting the current secondary key's
```

```

432 0495 3 | representative in the current primary data record.
433 0496 3
434 0497 3
435 0498 3 IF NOT RMSNULLKEY (.REC_ADDR)
436 0499 3 THEN
437 0500 3 LEAVE BLOCK;
438 0501 3
439 0502 3
440 0503 3
441 0504 3
442 0505 3
443 0506 3
444 0507 4 BEGIN
445 0508 4
446 0509 4 GLOBAL REGISTER
447 0510 4 R_BDB;
448 0511 4
449 0512 4 RMSRECORD_KEY (KEYBUF_ADDR(2));
450 0513 3 END;
451 0514 3
452 0515 3 | Position to and delete the SIDR array element pointing to the current
453 0516 3 primary data record for this secondary key from the file.
454 0517 3
455 0518 3 RMSDELETE_SIDR();
456 0519 2 END;
457 0520 2
458 0521 1 END;

```

```

.TITLE RM3DELETE
.IDENT \V04-000\

.EXTRN RMSCLEAN_BDB, RMSCSEARCH_TREE
.EXTRN RMSEXPNB KEYD, RMSEXT_ARRY RFA
.EXTRN RMSFIND_BY_ID, RMSGET_NEXT_KEY
.EXTRN RMSGETNXT_ARRY
.EXTRN RMSKEY_DESC, RMSMOVE
.EXTRN RMSNULKEY, RMSREC_OVHD
.EXTRN RMSRECORD_ID, RMSRECORD_KEY
.EXTRN RMSRECORD_VBN, RMSRLSBKT
.EXTRN RMSRU_JOURNAL, RMSSIDR END
.EXTRN RMSSIDR_FIRST, RMSUNPACK_REC
.EXTRN RMSUPDDELCOM

.PSECT RMSRMS$,$,NOWRT, GBL, PIC.2

```

54	DD	00000 RMSDEL_ALL_SIDR:			
04	04	38	0000G 30 00002 1\$:	PUSHL R4	0382
		A8	50 E9 00005	BSBW RMSGET_NEXT_KEY	0455
		F1	06 E1 00008	BLBC R0, 3\$	0471
22	22	1C	A7 E8 0000D	BBC #6, 4(RAB), 2\$	0473
		A7	08 AE B1 00011 2\$:	BLBS 28(IDX_DFN), 1\$	0481
			EA 1F 00016	CMPW RECORD_SIZE, 34(IDX_DFN)	
		56	68 A9 D0 00018	BLSSU 1\$	
			5C D4 0001C	MOVL 104(IRAB), REC_ADDR	0489
			56 DD 0001E	CLRL AP	0497
				PUSHL REC_ADDR	0498

RM3DELETE  
V04-000

RMSDEL\_ALL\_SIDR

H 11  
16-Sep-1984 01:42:30  
14-Sep-1984 13:01:19 VAX-11 Bliss-32 v4.0-742  
[RMS.SRC]RM3DELETE.B32:1

Page 10  
(2)

5E	0000G	30	00020	BSBW	RMSNULLKEY
D9	04	C0	00023	ADDL2	#4, SP
5C	50	E9	00026	BLBC	R0, 1\$
50	03	D0	00029	MOVL	#3, AP
	00B4	CA	3C 0002C	MOVZWL	180(IFAB), R0
	60	B940	9F 00031	PUSHAB	@96(IRAB)[R0]
	0000G	30	00035	BSBW	RMSRECORD_KEY
5E	04	C0	00038	ADDL2	#4, SP
	0000V	30	0003B	BSBW	RMSDELETE_SIDR
	C2	11	0003E	BRB	1\$
	10	BA	00040 3\$:	POPR	#^M<R4>
	05	00042		RSB	

; Routine Size: 67 bytes, Routine Base: RM\$RMS3 + 0000

#### CALLING SEQUENCE:

**RMSDELETE3B()**

**INPUT PARAMETERS:**

**NONE**

## IMPLICIT INPUTS:

IFAB

IFBSB\_NUM\_KEYS  
IFBSB\_PLG\_VER  
IFBSV\_RUP

- address of IFAB
- number of keys in the file
- prologue version of the file
- if set, Recovery Unit is in progress
- address of IRAB

```

517 0579 1 IRBSB_CUR_KREF - current positioning key of reference
518 0580 1 IRBSW_POS_ID - ID of positioning primary data record
519 0581 1 IRBSL_POS_VBN - VBN of positioning primary data record
520 0582 1 IRBSL_RECBUF - address of record buffer
521 0583 1 IRBSW_UDR_ID - ID of current primary data record
522 0584 1 IRBSL_UDR_VBN - VBN of current primary data record
523 0585 1
524 0586 1 OUTPUT PARAMETERS:
525 0587 1 NONE
526 0588 1
527 0589 1 IMPLICIT OUTPUTS:
528 0590 1
529 0591 1 IRAB
530 0592 1 IRBSV_FIND_LAST - 0, last operation was not a $FIND
531 0593 1 IRBSV_PUTS_LAST - 0, last operation was not a $PUT
532 0594 1 IRBSV_UPDATE - 0, last operation was not an $UPDATE
533 0595 1
534 0596 1 ROUTINE VALUE:
535 0597 1
536 0598 1 CUR - illegal or no current record
537 0599 1 RNL - current record not locked
538 0600 1 SUC - record successfully deleted
539 0601 1 various I/O errors
540 0602 1
541 0603 1
542 0604 1
543 0605 1 If record locking is unnecessary the record locks are not checked for.
544 0606 1 If automatic locking is not specified, the then the deleted record is
545 0607 1 not unlocked.
546 0608 1 If automatic locking is required, then the current primary data record
547 0609 1 is always unlocked, on success or failure.
548 0610 1 If the current process is within a Recovery Unit, and the file is being
549 0611 1 Recovery Unit Journalled, then the operation is RU Journalled
550 0612 1 before any permanent modification to the file takes place
551 0613 1 !--
552 0614 1
553 0615 2 BEGIN
554 0616 2
555 0617 2 BUILTIN
556 0618 2 AP;
557 0619 2
558 0620 2 EXTERNAL REGISTER
559 0621 2 COMMON_RAB_STR;
560 0622 2
561 0623 2 GLOBAL REGISTER
562 0624 2 COMMON_IO_STR,
563 0625 2 R_REC_ADDR_STR,
564 0626 2 R_IDX_DFN_STR;
565 0627 2
566 0628 2 LOCAL
567 0629 2 RECORD_SIZE;
568 0630 2
569 0631 2 | Perform checks common to both SUPDATE and SDELETE such as making sure
570 0632 2 | there is a current record and that it is locked, and then find the
571 0633 2 | current record by means of its RFA address. This will access both the
572 0634 2 | bucket containing the current record and the bucket containing the
573 0635 2 | current record's RRV, if it has one. The address of the BDB for the

```

```
574 0636 2 | current record bucket will be returned in IRBSL_CURBDB, and the address
575 0637 2 | of the BDB for the RRV bucket will be returned in IRBSL_NXTBDB.
576 0638 2
577 0639 2 | IRAB[IRBSV_UPDATE] = 0;
578 0640 2
579 0641 2 | RETURN_ON_ERROR (RMSUPDDELCOM());
580 0642 2
581 0643 2 | Retrieve the index descriptor for the primary key.
582 0644 2
583 0645 2 | RMSKEY_DESC (0);
584 0646 2
585 0647 2 | If the file contains alternate keys, then save the primary data record
586 0648 2 | (in unpacked format if the file's prologue version is 3), in a record
587 0649 2 | buffer so that the primary data record maybe deleted, and the record will
588 0650 2 | still available. This is so that the alternate keys maybe extracted from
589 0651 2 | it at a later time to be used in the deletion of the corresponding SIDRs.
590 0652 2
591 0653 2 | If the process is within a recovery unit and the file is being RU
592 0654 2 | Journalled, then unpack the primary data record regardless of whether or
593 0655 2 | not the file defines alternate keys.
594 0656 2
595 0657 2 | IF .IFAB[IFBSB_NUM_KEYS] GTRU 1
596 0658 2 | OR
597 0659 2 | .IFAB[IFBSV_RUP]
598 0660 2 | THEN
599 0661 3 | BEGIN
600 0662 3
601 0663 3
602 0664 3 | LOCAL
603 0665 3 | REC_SIZE,
604 0666 3 | SAVE_REC_ADDR : REF BBLOCK;
605 0667 3 | Retrieve the size of the current primary data record, and position
606 0668 3 | past the record overhead to the user data record itself.
607 0669 3
608 0670 3 | SAVE_REC_ADDR = .REC_ADDR;
609 0671 3 | REC_ADDR = .REC_ADDR + RM$REC_OVHD(0; REC_SIZE);
610 0672 3 | RECORD_SIZE = .REC_SIZE;
611 0673 3
612 0674 3 | If the file is a prologue 3 file, then the current primary data
613 0675 3 | record must be unpacked into the record buffer
614 0676 3
615 0677 3 | IF .IFBSB_PLG_VER] GEQU PLGSC_VER_3
616 0678 3 | THEN
617 0679 4 | BEGIN
618 0680 4
619 0681 4 | If the record is in a special format, then retrieve the true size
620 0682 4 | of the record from the last two bytes in the record's reserved
621 0683 4 | space.
622 0684 4
623 0685 4 | IF .SAVE_REC_ADDR[IRC$V_RU_UPDATE]
624 0686 4 | THEN
625 0687 5 | RECORD_SIZE = .(.REC_ADDR + .RECORD_SIZE
626 0688 4 | - IRC$C_DATSZFLD)<0,16>;
627 0689 4
628 0690 4 | As part of the process of unpacking the current primary data
629 0691 4 | record, RMS must extract the primary key from its position in
630 0692 4 | front of the rest of the data record, re-expand it if it is
```

631 0693 4 | compressed, and re-integrate it. If the current NRP positioning  
632 0694 4 | key of reference is the primary key, then when RMS positioned to  
633 0695 4 | the current primary data record it extracted its primary key into  
634 0696 4 | keybuffer 1 where it serves as part of the local NRP context. If  
635 0697 4 | this is indeed the case, then signal the data record unpacking  
636 0698 4 | routine that the primary key for this data record maybe found in  
637 0699 4 | keybuffer 1, and that there is no need to again extract and  
638 0700 4 | re-expand the primary key as part of the unpacking process;  
639 0701 4 | otherwise, signal that the entire unpacking process must be gone  
640 0702 4 | through.  
641 0703 4 |  
642 0704 4 | There is one case when RMS must signal that the entire unpacking  
643 0705 4 | process must be gone through even though the primary key is the  
644 0706 4 | current key of reference. This is when RMS positioned to the  
645 0707 4 | record by means of a random \$FIND. This type of operation does  
646 0708 4 | not update the NRP context.  
647 0709 4 |  
648 0710 5 IF (.IRAB[IRB\$B\_CUR\_KREF] EQLU 0)  
649 0711 4 | AND  
650 0712 5 (.IRAB[IRB\$W\_POS\_ID] EQLU .IRAB[IRB\$W\_UDR\_ID])  
651 0713 4 | AND  
652 0714 5 (.IRAB[IRB\$L\_POS\_VBN] EQLU .IRAB[IRB\$L\_UDR\_VBN])  
653 0715 4 | THEN  
654 0716 4 | | AP = 1  
655 0717 4 | ELSE  
656 0718 4 | | AP = 0;  
657 0719 4 |  
658 0720 4 | RECORD\_SIZE = RM\$UNPACK\_REC (.IRAB[IRB\$L\_RECBUF], .RECORD\_SIZE);  
659 0721 4 |  
660 0722 4 | If this file is being RU Journalled (Only Prologue 3 files are  
661 0723 4 | journalled), and the current process is within a Recovery Unit,  
662 0724 4 | then RU Journal the current operation and set the state bit  
663 0725 4 | IRB\$V RU\_DELETE so that the deletions are done such that no space  
664 0726 4 | at all is reclaimed.  
665 0727 4 |  
666 0728 4 | IF .IFAB[IFBSV\_RUP]  
667 0729 4 | THEN  
668 0730 5 | BEGIN  
669 0731 5 | REC\_ADDR = .IRAB[IRB\$L\_RECBUF];  
670 0732 5 | RETURN\_ON\_ERROR (RM\$RU\_JOURNAL\$ (RJRS\_DELETE,  
671 0733 5 | | .IRAB[IRB\$L\_UDR\_VBN],  
672 0734 5 | | .IRAB[IRB\$W\_UDR\_ID],  
673 0735 5 | | .RECORD\_SIZE),  
674 0736 5 | | RMSCLEAN\_BDB());  
675 0737 5 | | IRAB[IRBSV\_RU\_DELETE] = 1;  
676 0738 4 | | END;  
677 0739 4 | | END  
678 0740 4 |  
679 0741 4 | If the file is a prologue 1 or 2 file, then just move the primary data  
680 0742 4 | record into the record buffer.  
681 0743 4 |  
682 0744 3 | ELSE  
683 0745 3 | | RMSMOVE (.RECORD\_SIZE, .REC\_ADDR, .IRAB[IRB\$L\_RECBUF]);  
684 0746 3 |  
685 0747 3 | | Position back to the beginning of the primary data record - to the  
686 0748 3 | | first byte of the current primary data record's overhead.  
687 0749 3 |

```

688      0750 3      REC_ADDR = .SAVE_REC_ADDR;
689      0751 2      END;
690      0752 2
691      0753 2      | If the current record is not in its original bucket, process the RRV for
692      0754 2      | the current record. For prologue 3 files this involves deleting the RRV
693      0755 2      | entirely. For all other files, just the space occupied by the RRV pointer
694      0756 2      | to the current record is reclaimed. This means that the current record
695      0757 2      | can no longer be found through its secondary keys or by RFA access.
696      0758 2
697      0759 2      IF (BDB = .IRAB[IRB$L_NXTBDB]) NEQ 0
698      0760 2      THEN
699      0761 3      BEGIN
700      0762 3      IRAB[IRB$L_NXTBDB] = 0;
701      P 0763 3      RETURN_ON_ERROR (RMSDELETE_RRV(), BEGIN
702      P 0764 3      IRAB[IRB$V_RU_DELETE] = 0;
703      P 0765 3      RELEASE (IRAB[IRB$L_CURBDB]);
704      0766 3      END);
705      0767 2      END;
706      0768 2
707      0769 2      | Delete the current primary data record, mark the bucket dirty and release
708      0770 2      | it. If the current record's key is the high key in the primary data
709      0771 2      | bucket, then the current primary data record is just marked deleted.
710      0772 2
711      0773 2      BDB = .IRAB[IRB$L_CURBDB];
712      0774 2      IRAB[IRB$L_CURBDB] = 0;
713      0775 2
714      0776 2      RMSDELETE_UDR();
715      0777 2
716      0778 2      BDB[BDB$V_DRT] = 1;
717      0779 2      RETURN_ON_ERROR (RMSRLSBKT(0), IRAB[IRB$V_RU_DELETE] = 0);
718      0780 2
719      0781 2      | If the file contains alternate keys, delete all the SIDR entries for
720      0782 2      | the current record.
721      0783 2
722      0784 2      IF .IFAB[IFB$B_NUM_KEYS] GTRU 1
723      0785 2      THEN
724      0786 2      RMSDEL_ALL_SIDR (.RECORD_SIZE);
725      0787 2
726      0788 2      | Clear the IRB$V_RU_DELETE state bit regardless of whether this operation
727      0789 2      | was or wasn't RU Journalled, and then return success.
728      0790 2
729      0791 2      IRAB[IRB$V_RU_DELETE] = 0;
730      0792 2      RETURN RMSSUC7;
731      0793 1      END;

```

00FC 8F BB 00000 RMSDELETE3B::					
06	A9	08	8A 00004	PUSHR	#^M<R2,R3,R4,R5,R6,R7>
		0000G	30 00008	BICB2	#8, 6(IRAB)
03		50	E8 0000B	BSBW	RMSUPDDELCOM
		0101	31 0000E	BLBS	STATUS, 1\$
		7E	D4 00011 1\$:	BRW	13\$
		0000G	30 00013	CLRL	-(SP)
				BSBW	RMSKEY_DESC

: 0523  
: 0639  
: 0641  
: 0645  
:

		5E	01	00B2	04	C0	00016	ADDL2	#4, SP		0657	
					CA	91	00019	CMPB	178(IFAB), #1			
					09	1A	0001E	BGTRU	2\$			
		03	00A2	CA	02	E0	00020	BBS	#2, 162(IFAB), 2\$		0659	
					008A	31	00026	BRW	9\$			
				53	56	D0	00029	2\$:	MOVL	REC_ADDR, SAVE_REC_ADDR		0670
					51	D4	0002C	CLRL	R1		0671	
					0000G	30	0002E	BSBW	RMSREC OVHD			
				56	50	C0	00031	ADDL2	RO, REC_ADDR			
				52	51	D0	00034	MOVL	REC_SIZE, RECORD_SIZE		0672	
		03	00B7	CA	91	00037	CMPB	183(IFAB), #3			0677	
					65	1F	0003C	BLSSU	7\$			
		07	63	06	E1	0003E	BBC	#6, (SAVE_REC_ADDR), 3\$			0685	
					FE	A246	9F	PUSHAB	-2(RECORD_SIZE)[REC_ADDR]		0687	
			52	9E	3C	00042	MOVZWL	0(SP)+, RECORD_SIZE				
				00C3	C9	95	00046	3\$:	TSTB	195(IRAB)		0710
					17	12	0004D	BNEQ	4\$			
		00BC	C9	00BA	C9	B1	0004F	CMPW	186(IRAB), 188(IRAB)		0712	
					0E	12	00056	BNEQ	4\$			
		00B0	C9	00AC	C9	D1	00058	CMPL	172(IRAB), 176(IRAB)		0714	
					05	12	0005F	BNEQ	4\$			
			5C		01	D0	00061	MOVL	#1, AP		0716	
					02	11	00064	BRB	5\$			
					5C	D4	00066	CLRL	AP		0718	
			51		52	D0	00068	MOVL	RECORD_SIZE, R1		0720	
			50	68	A9	D0	0006B	MOVL	104(IRAB), RO			
		35	00A2	CA	0000G	30	0006F	BSBW	RMSUNPACK REC			
				52	50	D0	00072	MOVL	RO, RECORD_SIZE		0728	
			56	68	A9	D0	0007B	PUSHL	#2, 162(IFAB), 8\$		0731	
			7E	00BC	C9	3C	00081	MOVZWL	104(IRAB), REC_ADDR		0736	
				00B0	C9	DD	00086	PUSHL	RECORD_SIZE			
					05	DD	0008A	MOVZWL	188(IRAB), -(SP)			
					00000000G	EF	16	PUSHL	176(IRAB)			
			5E		10	C0	00092	JSB	#5			
			05		50	E8	00095	ADDL2	RMSRU JOURNAL3			
					0000G	30	00098	BLBS	#16, SP			
					75	11	0009B	BSBW	STATUS, 6\$			
		07	A9		20	88	0009D	BRB	RMSCLEAN_BDB			
					0D	11	000A1	BISB2	13\$			
					68	A9	DD	#32, 7(IRAB)			0737	
					0044	8F	BB	BRB	8\$		0677	
							000A3	PUSHL	104(IRAB)		0745	
							7\$:	PUSHR	#^M<R2,R6>			
			5E		0000G	30	000AA	BSBW	RMSMOVE			
			56		OC	C0	000AD	ADDL2	#12, SP			
			54		53	D0	000B0	MOVL	SAVE_REC_ADDR, REC_ADDR		0750	
				3C	A9	D0	000B3	MOVL	60(IRAB), BDB		0759	
					24	13	000B7	BEQL	10\$			
					3C	A9	D4	CLRL	60(IRAB)		0762	
						0000V	30	BSBW	RMSDELETE RRV		0766	
			51		50	D0	000BF	MOVL	RO, STATUS			
			18		51	E8	000C2	BLBS	STATUS, 10\$			
		07	A9		20	8A	000C5	BICB2	#32, 7(IRAB)			
			54		20	A9	D0	MOVL	32(IRAB), BDB			
					20	A9	D4	CLRL	32(IRAB)			
					7E	D4	000CD	CLRL	-(SP)			
					0000G	30	000D2	BSBW	RMSRLSBKT			

5E	04	C0	000D5	ADDL2	#4, SP	:	
50	51	D0	000D8	MOVL	STATUS, R0	:	
54	20	35	11 000DB	BRB	13\$	0773	
	20	A9	D0 000DD	MOVL	32(IRAB), BDB	0774	
		A9	D4 000E1	CLRL	32(IRAB)	0776	
0A	A4	0000V	30 000E4	BSBW	RMSDELETE UDR	0778	
		02	88 000E7	BISB2	#2, 10(BDB)	0779	
		7E	D4 000EB	CLRL	-(SP)		
		0000G	30 000ED	BSBW	RMSRLSBKT		
5E	04	C0	000F0	ADDL2	#4, SP		
06	50	E8	000F3	BLBS	STATUS, 11\$		
07	A9	20	8A 000F6	BICB2	#32, 7(IRAB)		
		16	11 000FA	BRB	13\$	0784	
01	00B2	CA	91 000FC	11\$:	CMPB	178(IFAB), #1	
		08	1B 00101	BLEQU	12\$		
		52	DD 00103	PUSHL	RECORD_SIZE	0786	
		FEB5	30 00105	BSBW	RMSDEL_ALL_SIDR		
07	5E	04	C0 00108	ADDL2	#4, SP		
	A9	20	8A 0010B	12\$:	BICB2	#32, 7(IRAB)	0791
	50	01	D0 0010F	MOVL	#1, R0	0792	
	00FC	8F	BA 00112	13\$:	POPR	#^M<R2,R3,R4,R5,R6,R7>	0793
			05 00116	RSB			

: Routine Size: 279 bytes, Routine Base: RMSRMS3 + 0043

0794 1 %SBTTL 'RMSDELETE RRV'  
0795 1 GLOBAL ROUTINE RMSDELETE\_RRV : RL\$RABREG\_4567 =  
0796 1 ++  
0797 1  
0798 1  
0799 1  
0800 1  
0801 1  
0802 1  
0803 1  
0804 1  
0805 1  
0806 1  
0807 1  
0808 1  
0809 1  
0810 1  
0811 1  
0812 1  
0813 1  
0814 1  
0815 1  
0816 1  
0817 1  
0818 1  
0819 1  
0820 1  
0821 1  
0822 1  
0823 1  
0824 1  
0825 1  
0826 1  
0827 1  
0828 1  
0829 1  
0830 1  
0831 1  
0832 1  
0833 1  
0834 1  
0835 1  
0836 1  
0837 1  
0838 1  
0839 1  
0840 1  
0841 1  
0842 1  
0843 1  
0844 1  
0845 1  
0846 1  
0847 1  
0848 1  
0849 1  
0850 2

FUNCTIONAL DESCRIPTION:  
Delete the RRV for the current primary data record. If the file is a prologue 3 file the RRV is entirely deleted; otherwise, it is marked deleted and just the space occupied by the pointer is reclaimed.  
If the state bit IRB\$V\_RU\_DELETE is set, the RRV is just marked RU\_DELETE. Likewise, if the state bit IRB\$V\_RU\_UNDEL is set, then the RU\_DELETE bit in the RRV;s control byte is cleared.

CALLING SEQUENCE:  
RMSDELETE\_RRV()

INPUT PARAMETERS:  
NONE

IMPLICIT INPUTS:  
BDB BDB\$L\_ADDR - BDB of buffer with RRV bucket in it  
- address of buffer  
IFAB IFB\$B\_PLG\_VER - address of IFAB  
- prologue version of file  
IRAB IRB\$V\_RU\_DELETE - if set, mark RU\_DELETE and do not reclaim  
IRB\$V\_RU\_UNDEL - if set, un-delete the RRV  
REC\_ADDR - address of record whose RRV is to be deleted

OUTPUT PARAMETERS:  
NONE

IMPLICIT OUTPUTS:  
IDX\_DFN - index descriptor for the primary key

ROUTINE VALUE:  
Value of RLSBKT when writing out bucket with RRV deleted

SIDE EFFECTS:  
AP destroyed.  
IDX\_DFN is set up for the primary key.  
The freespace offset in the RRV bucket is updated to reflect the amount of space reclaimed.

--  
BEGIN

```
790 0851 2
791 0852 2
792 0853 2
793 0854 2
794 0855 2
795 0856 2
796 0857 2
797 0858 2
798 0859 2
799 0860 2
800 0861 2
801 0862 2
802 0863 2
803 0864 2
804 0865 2
805 0866 2
806 0867 2
807 0868 2
808 0869 2
809 0870 2
810 0871 2
811 0872 2
812 0873 2
813 0874 2
814 0875 2
815 0876 2
816 0877 2
817 0878 2
818 0879 2
819 0880 2
820 0881 2
821 0882 2
822 0883 2
823 0884 2
824 0885 2
825 0886 2
826 0887 2
827 0888 2
828 0889 2
829 0890 2
830 0891 2
831 0892 2
832 0893 2
833 0894 2
834 0895 2
835 0896 2
836 0897 2
837 0898 2
838 0899 2
839 0900 2
840 0901 2
841 0902 2
842 0903 2
843 0904 2
844 0905 2
845 0906 2
846 0907 3

    BUILTIN
        AP;

    EXTERNAL REGISTER
        R_BDB_STR,
        COMMON_RAB_STR,
        R_IDX_DFN_STR,
        R_REC_ADDR_STR;

    GLOBAL REGISTER
        R_BKT_ADDR_STR;

    LOCAL
        DEL_RRV_SIZE,
        LENGTH,
        RRV_SIZE,
        SAVE_REC_ADDR;

    ! Obtain the key descriptor for the primary key of reference.
    RMSKEY_DESC(0);

    SAVE_REC_ADDR = .REC_ADDR;

    ! Extract the RRV ID of the current primary data record.
    AP = RMSRECORD_ID();

    ! Position to the RRV to be deleted, the RRV for the current primary data
    ! record. It is impossible for this positioning to fail as long as the
    ! bucket containing the RRV has not been released since RMSFIND_BY_RRV
    ! accessed it.

    BKT_ADDR = .BDB[BDB$L_ADDR];
    RMSFIND_BY_ID();

    ! If it is indicated that the RRV should just be marked RU_DELETE and that
    ! no space should be reclaimed, then do so by setting the RU_DELETE bit
    ! within the RRV's control byte.

    IF .IRAB[IRB$V_RU_DELETE]
    THEN
        REC_ADDR[IRC$V_RU_DELETE] = 1

    ! If it is indicated that the RRV should be un-deleted, then do so by
    ! clearing the RU_DELETE bit in the RRV's control byte.

    ELSE
        IF .IRAB[IRB$V_RU_UNDEL]
        THEN
            REC_ADDR[IRC$V_RU_DELETE] = 0

        ! Delete the RRV reclaiming as much space as is possible.

        ELSE
            BEGIN
```

```
847 0908 3
848 0909 3
849 0910 3
850 0911 3
851 0912 3
852 0913 3
853 0914 4
854 0915 4
855 0916 4
856 0917 4
857 0918 3
858 0919 4
859 0920 4
860 0921 4
861 0922 3
862 0923 3
863 0924 3
864 0925 3
865 0926 3
866 0927 4
867 0928 3
868 0929 3
869 0930 3
870 0931 3
871 0932 3
872 0933 3
873 0934 3
874 0935 3
875 0936 3
876 0937 3
877 0938 3
878 0939 3
879 0940 3
880 0941 3
881 0942 3
882 0943 3
883 0944 3
884 0945 3
885 0946 3
886 0947 3
887 0948 3
888 0949 3
889 0950 2
890 0951 2
891 0952 2
892 0953 2
893 0954 2
894 0955 2
895 0956 2
896 0957 2
897 0958 2
898 0959 2
899 0960 1

  ! Setup a series of constants to be used in deleting the RRV. These
  ! constants are prologue dependent.

  IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
  THEN
    BEGIN
      RRV_SIZE = IRC$C_FIXOVHDSZ;
      DEL_RRV_SIZE = 2;
    END
  ELSE
    BEGIN
      RRV_SIZE = IRC$C_FIXOVHSZ;
      DEL_RRV_SIZE = 0;
    END;

  ! Delete/Squish the current primary data record's RRV and fix up
  ! the RRV bucket's freespace.

  LENGTH = (.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
            - (.REC_ADDR + .RRV_SIZE);

  IF .LENGTH GTRU 0
  THEN
    RM$MOVE (.LENGTH,
              .REC_ADDR + .RRV_SIZE,
              .REC_ADDR + .DEL_RRV_SIZE);

  BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
                            - .RRV_SIZE + .DEL_RRV_SIZE;

  ! If the file is not a prologue 3 file, then the RRV of the current
  ! primary data record was just squished. The RRV pointer was
  ! removed, but the control byte and record ID fields remain. In
  ! this case RMS wants to setup the control byte of the squished RRV
  ! to indicate that it has been deleted, is an RRV, and doesn't
  ! contain a pointer.

  IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
  THEN
    REC_ADDR[IRC$B_CONTROL] = IRC$M_RRV OR IRC$M_DELETED
                               OR IRC$M_NOPTRSZ;
  END;

  ! Restore the address of the current primary data record and release the
  ! RRV's bucket after marking it dirty.

  REC_ADDR = .SAVE_REC_ADDR;

  BDB[BDB$V_DRT] = 1;
  RETURN RM$RLSBKT (RLSSM_WRT_THRU)

  END;
```

2C BB 00000 RMSDELETE RRV::					
		5E	08 C2 00002	POSHR #^M<R2,R3,R5>	0795
			7E D4 00005	SUBL2 #8, SP	
			0000G 30 00007	CLRL -(SP)	0872
		5E	04 C0 0000A	BSBW RMSKEY_DESC	
		6E	56 D0 0000D	ADDL2 #4, SP	0874
			0000G 30 00010	MOVL REC_ADDR, SAVE_REC_ADDR	
		5C	50 D0 00013	BSBW RMSRECORD_ID	0878
		55	18 A4 D0 00016	MOVL R0, AP	
			0000G 30 0001A	MOVL 24(BDB), BKT_ADDR	0885
05	07	A9	05 E1 0001D	BSBW RMSFIND_BY_ID	
		66	20 88 00022	BBC #5 7(IRAB), 1\$	0886
			53 11 00025	BISB2 #32, (REC_ADDR)	0892
05	07	A9	06 E1 00027	BRB 6\$	
		66	20 8A 0002C	BBC #6 7(IRAB), 2\$	0894
		03	0087 49 11 0002F	BICB2 #32, (REC_ADDR)	
			CA 91 00031	BRB 6\$	0900
			09 1E 00036	CMPB 183(IFAB), #3	
		04	53 07 D0 00038	BGEQU 3\$	0912
		AE	02 D0 0003B	MOVL #7, RRV_SIZE	
			06 11 0003F	MOVL #2, DEL_RRV_SIZE	0915
		53	09 D0 00041	BRB 4\$	
			04 AE D4 00044	MOV L #9, RRV_SIZE	0916
		52	04 A5 3C 00047	CLRL DEL_RRV_SIZE	
51	50		55 52 C1 0004B	MOVZWL 4(BRT_ADDR), R2	0921
			56 53 C1 0004F	ADDL3 R2, BRT_ADDR, R1	
			51 50 C2 00053	ADDL3 RRV_SIZE, REC_ADDR, R0	0927
			0E 13 00056	SUBL2 R0, LENGTH	
			04 BE46 9F 00058	BEQL 5\$	0928
			50 DD 0005C	PUSHAB @DEL_RRV_SIZE[REC_ADDR]	
			51 DD 0005E	PUSHL R0	0930
			0000G 30 00060	PUSHL LENGTH	
		5E	0C C0 00063	BSBW RMSMOVE	0934
04	50	52	53 C3 00066	ADDL2 #12, SP	
		50	04 AE A1 0006A	SUBL3 RRV_SIZE, R2, R0	0937
		03	0087 CA 91 00070	ADDW3 DEL_RRV_SIZE, R0, 4(BKT_ADDR)	
			03 1E 00075	CMPB 183(IFAB), #3	0946
		66	1C 90 00077	BGEQU 6\$	
		56	6E D0 0007A	MOVB #28, (REC_ADDR)	0949
0A	A4	02	02 88 0007D	MOVL SAVREC_ADDR, REC_ADDR	
			02 DD 00081	BISB2 #2, TO(BDB)	0955
		5E	0000G 30 00083	PUSHL #2	
			0C C0 00086	BSBW RM\$RLSBKT	0957
			2C BA 00089	ADDL2 #12, SP	
			05 0008B	POPR #^M<R2,R3,R5>	0960
				RSB	

: Routine Size: 140 bytes, Routine Base: RM\$RMS3 + 015A

```
901 0961 1 %SBTTL 'RMSDELETE SIDR'  
902 0962 1 GLOBAL ROUTINE RM$DELETE_SIDR : RL$RABREG_7 =  
903 0963 1  
904 0964 1 !++  
905 0965 1  
906 0966 1 FUNCTIONAL DESCRIPTION:  
907 0967 1  
908 0968 1 This routine's responsibility is to position to the SIDR array element  
909 0969 1 pointing to the current primary data record for a given key of  
910 0970 1 reference and delete it. The secondary key in keybuffer 2, and the  
911 0971 1 RFA address of the current primary data record, found as part of the  
912 0972 1 local NRP context in the IRAB, are utilized in this positioning.  
913 0973 1 Deletion of the appropriate SIDR array element consists of one of the  
914 0974 1 following:  
915 0975 1  
916 0976 1 1. Removal of the entire SIDR if duplicates are not allowed.  
917 0977 1  
918 0978 1 2. Marking the SIDR array element as deleted and not recovering any  
919 0979 1 space if duplicates are allowed for this key of reference and the  
920 0980 1 file is a prologue 1 or 2 file.  
921 0981 1  
922 0982 1 3. Marking the SIDR array element as deleted and not recovering any  
923 0983 1 space if duplicates are allowed for this key of reference, the file  
924 0984 1 is a prologue 3 file, and the element is the first element in the SIDR  
925 0985 1 array.  
926 0986 1  
927 0987 1 4. Marking the SIDR element deleted and squishing out the space  
928 0988 1 occupied by the RRV pointer if duplicates are allowed for this key  
929 0989 1 of reference, the file is a prologue 3 file, and the element is not  
930 0990 1 the first element in the SIDR array.  
931 0991 1  
932 0992 1 5. Removal of the entire SIDR array if duplicates are allowed, this is  
933 0993 1 the first SIDR with this key value, the SIDR is not the physically  
934 0994 1 last SIDR in the bucket, and every single element within the SIDR  
935 0995 1 array has been deleted.  
936 0996 1  
937 0997 1 CALLING SEQUENCE:  
938 0998 1  
939 0999 1 RM$DELETE_SIDR()  
940 1000 1  
941 1001 1 INPUT PARAMETERS:  
942 1002 1 NONE  
943 1003 1  
944 1004 1 IMPLICIT INPUTS:  
945 1005 1  
946 1006 1 IDX_DFN - address of index descriptor  
947 1007 1 IDX$B_KEYSZ - size of alternate key  
948 1008 1  
949 1009 1 IRAB - address of IRAB  
950 1010 1 IRBSW_UDR_ID - RFA VBN of the current primary data record  
951 1011 1 IRBSL_UDR_VBN - RFA ID of the current primary data record  
952 1012 1  
953 1013 1 OUTPUT PARAMETERS:  
954 1014 1 NONE  
955 1015 1  
956 1016 1 IMPLICIT OUTPUTS:  
957 1017 1
```

958 1018 1 IRAB  
959 1019 1 IRB\$B\_KEYSZ - address of IRAB  
960 1020 1 IRB\$B\_STOPLEVEL - size of alternate key for key of reference  
961 1021 1 - level of index to search to (set to 0)  
962 1022 1 ROUTINE VALUE:  
963 1023 1 Status of the RMSRLSBKT call (success or error) that released  
964 1024 1 the modified bucket.  
965 1025 1 BUG - if the SIDR array element could not be located  
966 1026 1  
967 1027 1 SIDE EFFECTS:  
968 1028 1 Modified bucket is released.  
969 1029 1 IRBSV\_POSDELETE set within IRBSW\_SRCHFLGS.  
970 1030 1  
971 1031 1  
972 1032 1  
973 1033 1 ---  
974 1034 1  
975 1035 2 BEGIN  
976 1036 2  
977 1037 2 EXTERNAL REGISTER  
978 1038 2 COMMON RAB\_STR,  
979 1039 2 R\_IDX\_DFN\_STR;  
980 1040 2  
981 1041 2 GLOBAL REGISTER  
982 1042 2 COMMON IO\_STR,  
983 1043 2 R\_REC\_ADDR\_STR;  
984 1044 2  
985 1045 2 LABEL  
986 1046 2 FIND\_ELEMENT;  
987 1047 2  
988 1048 2 LOCAL  
989 1049 2 BEGIN\_OF\_SIDR;  
990 1050 2  
991 1051 2 | Since RMS is going to position so it can delete a SIDR array element,  
992 1052 2 | set the appropriate search flag, and make sure the key size is set up.  
993 1053 2  
994 1054 2 IRAB[IRB\$B\_STOPLEVEL] = 0;  
995 1055 2 IRAB[IRBSW\_SRCHFLGS] = IRBSM\_POSDELETE;  
996 1056 2 IRAB[IRB\$B\_KEYSZ] = .IDX\_DFN[IDX\$B\_KEYSZ];  
997 1057 2  
998 1058 2 | Position to the SIDR array element pointing to the current primary data  
999 1059 2 | record for this key of reference. This loop will only be exited either  
1000 1060 2 | when the array element has been located or all SIDR elements with this  
1001 1061 2 | key value are exhausted.  
1002 1062 2  
1003 1063 2 FIND\_ELEMENT:  
1004 1064 3 BEGIN  
1005 1065 3  
1006 1066 3 LOCAL  
1007 1067 3 END\_OF\_SIDR,  
1008 1068 3 ID,  
1009 1069 3 STATUS,  
1010 1070 3 VBN;  
1011 1071 3  
1012 1072 3 WHILE 1  
1013 1073 3 DO  
1014 1074 4 BEGIN

```
1015 1075 4
1016 1076 4
1017 1077 4
1018 1078 4
1019 1079 4
1020 1080 4
1021 1081 5
1022 1082 4
1023 1083 5
1024 1084 5
1025 1085 5
1026 1086 4
1027 1087 4
1028 1088 4
1029 1089 4
1030 1090 4
1031 1091 4
1032 1092 4
1033 1093 4
1034 1094 4
1035 1095 4
1036 1096 4
1037 1097 4
1038 1098 4
1039 1099 4
1040 1100 4
1041 1101 4
1042 1102 4
1043 1103 4
1044 1104 4
1045 1105 4
1046 1106 4
1047 1107 4
1048 1108 4
1049 1109 4
1050 1110 5
1051 1111 4
1052 1112 5
1053 1113 4
1054 1114 4
1055 1115 4
1056 1116 4
1057 1117 4
1058 1118 4
1059 1119 4
1060 1120 4
1061 1121 4
1062 1122 3
1063 1123 2
1064 1124 2
1065 1125 2
1066 1126 2
1067 1127 2
1068 1128 2
1069 1129 2
1070 1130 2
1071 1131 2

1076 4
1077 4
1078 4
1079 4
1080 4
1081 5
1082 4
1083 5
1084 5
1085 5
1086 4
1087 4
1088 4
1089 4
1090 4
1091 4
1092 4
1093 4
1094 4
1095 4
1096 4
1097 4
1098 4
1099 4
1100 4
1101 4
1102 4
1103 4
1104 4
1105 4
1106 4
1107 4
1108 4
1109 4
1110 5
1111 4
1112 5
1113 4
1114 4
1115 4
1116 4
1117 4
1118 4
1119 4
1120 4
1121 4
1122 3
1123 2
1124 2
1125 2
1126 2
1127 2
1128 2
1129 2
1130 2
1131 2

| If RMS is unable to find an array element pointing to the current
| primary data record, then something is very wrong. Return an internal
| bug error, and save the status from RM$CSEARCH_TREE, in the RABs STV
| field.

| IF NOT (STATUS = RM$CSEARCH_TREE())
| THEN
| BEGIN
|   RAB[RAB$L_STV] = .STATUS;
|   RETURN RM$ERR(BUG);
| END;

| Prepare to search the SIDR array for the element pointing to the
| current primary data record.

| BEGIN_OF_SIDR = .REC_ADDR;
| END_OF_SIDR = RM$SIDR_END();

| Position to the first array element in the SIDR array.

| REC_ADDR = RM$SIDR_FIRST(0);

| Search the current SIDR array for the element corresponding to the
| current primary data record.

WHILE .REC_ADDR LSSA .END_OF_SIDR
DO

  | If after extracting out the RFA pointer from the current SIDR
  | array element, RMS finds that it does indeed point to the
  | current primary data record, then exit the search loop

  IF RM$EXT_ARRY_RFA (VBN, ID)
    AND
    (.IRAB[IRBSW_UDR_ID] EQLU .ID)
    AND
    (.IRAB[IRB$L_UDR_VBN] EQLU .VBN)
  THEN
    LEAVE FIND_ELEMENT

  | If the current array element is deleted or does not point to the
  | current primary data record then proceed to the next element in
  | the SIDR array.

  ELSE
    RM$GETNXT_ARRAY();

  END;
END;

| Delete the SIDR array pointing to the current primary data record
| for this key of reference. The deletion rules are stated above.

BDB = .IRAB[IRB$L_CURBDB];
IRAB[IRB$L_CURBDB] = 0;
BKT_ADDR = .BDB[BDB$L_ADDR];
```

```

: 1072 1132 2 RM$SQUISH_SIDR (0, .BEGIN_OF_SIDR);
: 1073 1133 2
: 1074 1134 2 ! Mark the bucket dirty, and release it.
: 1075 1135 2
: 1076 1136 2 BDB[BDB$V_DRT] = 1;
: 1077 1137 2 RETURN RM$RLSBKT(0);
: 1078 1138 2
: 1079 1139 1 END;

```

				007C 8F BB 00000 RMSDELETE SIDR::	
				PUSHR #^M<R2,R3,R4,R5,R6>	0962
			5E	SUBL2 #8, SP	
			41	CLRB 65(IRAB)	1054
			A9	MOVW #4, 66(IRAB)	1055
			20	MOVB 32(IDX DFN), 166(IRAB)	1056
			C9	BSBW RM\$CSEARCH_TREE	1081
			54	MOVL R0, STATUS	
			0B	BLBS STATUS, 2\$	
			A8	MOVL STATUS, 12(RAB)	1084
			50	MOVZWL #33844, R0	1085
			8434	BRB 6\$	
			53	MOVL REC_ADDR, BEGIN_OF_SIDR	1091
			55	BSBW RM\$SIDR_END	1092
			7E	MOVL R0, END_OF_SIDR	
			0000G	CLRL -(SP)	1096
			54	BSBW RM\$SIDR_FIRST	
			0B	ADDL2 #4, SP	
			A8	MOVL R0, REC_ADDR	
			55	CMPL REC_ADDR, END_OF_SIDR	1101
			56	BGEQU 1\$	
			5E	PUSHL SP	1108
			56	PUSHAB VBN	
			5E	BSBW RM\$EXT_ARRY_RFA	
			11	ADDL2 #8, SP	
			10	BLBC R0, 4\$	
			08	CMPZV #0, #16, 188(IRAB), ID	1110
			AE	BNEQ 4\$	
			04	CMPL 176(IRAB), VBN	1112
			AE	BEQL 5\$	
			00B0	BSBW RM\$GETNXT_ARRAY	1121
			C9	BRB 3\$	1108
			D1	MOVL 32(IRAB), BDB	1128
			05	CLRL 32(IRAB)	1129
			13	MOVL 24(BDB), BKT ADDR	1131
			0000G	PUSHL BEGIN_OF_SIDR	1132
			30	CLRL -(SP)	
			00060	BSBW RM\$SQUISH_SIDR	
			4\$:	ADDL2 #4, SP	
			D7	BISB2 #2, 10(BDB)	1136
			11	CLRL (SP)	1137
			00063	BSBW RM\$RLSBKT	
			20	ADDL2 #4, SP	
			A9	ADDL2 #8, SP	1139
			00065		
			54		
			20		
			A9		
			D4		
			00069		
			55		
			18		
			A4		
			0006C		
			53		
			DD		
			00070		
			7E		
			D4		
			00072		
			0000V		
			30		
			00074		
			04		
			C0		
			00077		
			02		
			88		
			0007A		
			6E		
			D4		
			0007E		
			0000G		
			30		
			00080		
			04		
			C0		
			00083		
			08		
			C0		
			00086		
			6\$:		

RM3DELETE  
V04-000

RMSDELETE\_SIDR

K 12

16-Sep-1984 01:42:30 VAX-11 Bliss-32 V4.0-742  
14-Sep-1984 13:01:19 [RMS.SRC]RM3DELETE.B32;1

Page 26  
(5)

007C 8F BA 00089  
05 0008D      POPR      #^M<R2,R3,R4,R5,R6>  
                  RSB

: Routine Size: 142 bytes,    Routine Base: RM\$RMS3 + 01E6

```

1081 1140 1 %SBTTL 'RMSDELETE UDR'
1082 1141 1 GLOBAL ROUTINE RMSDELETE_UDR : RL$RABREG_4567 NOVALUE =
1083 1142 1
1084 1143 1 ++
1085 1144 1
1086 1145 1 FUNCTIONAL DESCRIPTION:
1087 1146 1
1088 1147 1 This routine's responsibility is the deletion of a primary data record.
1089 1148 1 Most but not all of the time, the record being deleted is the current
1090 1149 1 primary data record. The rules for how primary data records are deleted
1091 1150 1 are as follows:
1092 1151 1
1093 1152 1 1. If the primary data record is marked deleted, then the entire record
1094 1153 1 is always deleted.
1095 1154 1
1096 1155 1 2. If duplicate primary keys are not allowed, and the record is not the
1097 1156 1 last primary data record in the bucket then the entire primary data
1098 1157 1 record is deleted.
1099 1158 1
1100 1159 1 3. If duplicate primary keys are not allowed, and the record is the
1101 1160 1 last primary data record in the bucket then the primary data record
1102 1161 1 is marked deleted, and the space occupied by the data portion of the
1103 1162 1 record is reclaimed if the file's prologue version is 3.
1104 1163 1
1105 1164 1 4. If duplicate primary keys are allowed then the primary data record
1106 1165 1 is marked deleted, and the space occupied by the data portion of the
1107 1166 1 record is recovered if the file's prologue version is 3.
1108 1167 1
1109 1168 1 5. If the state bit IRB$V_RU_DELETE is set, then the primary data
1110 1169 1 record is just marked RU_DELETE and no space is reclaimed.
1111 1170 1
1112 1171 1 6. If the state bit IRB$V_RU_UNDEL is set, then the primary data record
1113 1172 1 is un-deleted by clearing the RU_DELETE bit within the record control
1114 1173 1 byte.
1115 1174 1
1116 1175 1 7. If the primary data record is completely deleted, the record was in
1117 1176 1 its original bucket (ie - a RRV does not exist), and the file's
1118 1177 1 prologue version is 1 or 2, then a two-byte RRV is created at the
1119 1178 1 end of the bucket for this record to prevent its ID from being
1120 1179 1 recycled.
1121 1180 1
1122 1181 1 CALLING SEQUENCE:
1123 1182 1
1124 1183 1 RMSDELETE_UDR()
1125 1184 1
1126 1185 1 INPUT PARAMETERS:
1127 1186 1 NONE
1128 1187 1
1129 1188 1 IMPLICIT INPUTS:
1130 1189 1
1131 1190 1 BDB
1132 1191 1 BDB$L_ADDR
1133 1192 1 BDB$L_VBN
1134 1193 1
1135 1194 1 IDX_DFN
1136 1195 1 IDX$V_DUPKEYS
1137 1196 1 IDX$V_KEY_COMPR

```

1138 1197 1  
1139 1198 1  
1140 1199 1  
1141 1200 1  
1142 1201 1  
1143 1202 1  
1144 1203 1  
1145 1204 1  
1146 1205 1  
1147 1206 1  
1148 1207 1  
1149 1208 1  
1150 1209 1  
1151 1210 1  
1152 1211 1  
1153 1212 1  
1154 1213 1  
1155 1214 1  
1156 1215 1  
1157 1216 1  
1158 1217 1  
1159 1218 1  
1160 1219 1  
1161 1220 1  
1162 1221 1  
1163 1222 1  
1164 1223 1  
1165 1224 1  
1166 1225 1  
1167 1226 1  
1168 1227 1  
1169 1228 1  
1170 1229 1  
1171 1230 1  
1172 1231 1  
1173 1232 1  
1174 1233 1  
1175 1234 1  
1176 1235 1  
1177 1236 2  
1178 1237 2  
1179 1238 2  
1180 1239 2  
1181 1240 2  
1182 1241 2  
1183 1242 2  
1184 1243 2  
1185 1244 2  
1186 1245 2  
1187 1246 2  
1188 1247 2  
1189 1248 2  
1190 1249 2  
1191 1250 2  
1192 1251 2  
1193 1252 2  
1194 1253 2

IFAB IFBSW\_KBUFSZ  
IFBSB\_PLG\_VER

IRAB IRBSL\_KEYBUF  
IRBSV\_RU\_DELETE  
IRBSV\_RU\_UNDEL

REC\_ADDR

OUTPUT PARAMETERS:  
NONE

IMPLICIT OUTPUTS:  
NONE

ROUTINE VALUE:  
NONE

SIDE EFFECTS:  
AP is trashed.  
Keybuffer 5 is trashed (if the primary key of the following primary data record had to be re-expanded).  
The freespace offset in the bucket is updated to reflect the amount of space reclaimed.  
REC\_ADDR is unchanged. It either points to the deleted record if the target primary data record could not be completely removed, or it points to whatever followed the deleted primary data record (if anything) if it could.  
If this is a prologue 1 or 2 file, and the primary data record which was deleted is in its original bucket, then a two-byte RRV is created to replace the deleted primary data record, provided the space occupied by the record was completely recovered.

--

BEGIN

BUILTIN AP;

EXTERNAL REGISTER  
R\_BDB\_STR,  
COMMON\_RAB\_STR,  
R\_IDX\_DFN\_STR,  
R\_REC\_ADDR\_STR;

GLOBAL REGISTER  
R\_BKT\_ADDR\_STR;

FIELD  
DELETE\_FLAGS =  
SET  
BUILD\_RRV = [0,0,1,0],

```
1195 1254 2      LAST_RECORD      = [0,1,1,0],  
1196 1255 2      RE_EXPAND_KEY    = [0,2,1,0],  
1197 1256 2      TES;  
1198 1257 2  
1199 1258 2      LOCAL  
1200 1259 2      END_OF_BUCKET    : REF BBLOCK,  
1201 1260 2      FLAGS          : BLOCK[1,BYTE]  
1202 1261 2      FIELD(DELETE_FLAGS),  
1203 1262 2      NEXT_REC_ADDR  : REF BBLOCK,  
1204 1263 2      REC_OVHD;  
1205 1264 2  
1206 1265 2      | If it is indicated that the primary data record should just be marked  
1207 1266 2      RU_DELETE and that no space should be reclaimed, then do so by setting  
1208 1267 2      the RU_DELETE bit within the RRV's control byte.  
1209 1268 2  
1210 1269 2      IF .IRAB[IRBSV_RU_DELETE]  
1211 1270 2      THEN  
1212 1271 3      BEGIN  
1213 1272 3      REC_ADDR[IRC$V_RU_DELETE] = 1;  
1214 1273 3      RETURN;  
1215 1274 3      END  
1216 1275 3  
1217 1276 3      | If it is indicated that the primary data record should be un-deleted,  
1218 1277 3      then do so by clearing the RU_DELETE bit in the primary data record's  
1219 1278 3      control byte.  
1220 1279 3  
1221 1280 2      ELSE  
1222 1281 2      IF .IRAB[IRBSV_RU_UNDEL]  
1223 1282 2      THEN  
1224 1283 3      BEGIN  
1225 1284 3      REC_ADDR[IRC$V_RU_DELETE] = 0;  
1226 1285 3      RETURN;  
1227 1286 2      END;  
1228 1287 2  
1229 1288 2      | Obtain the address of the primary data bucket, and compute the first  
1230 1289 2      free byte in the data bucket.  
1231 1290 2  
1232 1291 2      FLAGS = 0;  
1233 1292 2      BKT_ADDR = .BDB[BDB$L_ADDR];  
1234 1293 2      END_OF_BUCKET = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];  
1235 1294 2  
1236 1295 2      | Obtain the overhead for ALL records in this primary data bucket, and  
1237 1296 2      compute the address of the first primary data record which would follow  
1238 1297 2      the primary data record to be deleted.  
1239 1298 2  
1240 1299 3      BEGIN  
1241 1300 3  
1242 1301 3      LOCAL  
1243 1302 3      REC_SIZE;  
1244 1303 3  
1245 1304 3      REC_OVHD = RM$REC_OVHD(0; REC_SIZE);  
1246 1305 3      NEXT_REC_ADDR = .REC_ADDR + .REC_OVHD + .REC_SIZE;  
1247 1306 2      END;  
1248 1307 2  
1249 1308 2      | Determine whether the primary data record to be deleted is the last  
1250 1309 2      record in the bucket, and set the local state flag accordingly.  
1251 1310 2
```

```
: 1252      1311 3  IF (.NEXT_REC_ADDR EQLA .END_OF_BUCKET)
: 1253      1312 2  OR
: 1254      1313 2  .NEXT_REC_ADDR[IRC$V_RRV]
: 1255      1314 2  THEN
: 1256      1315 2  FLAGS[LAST_RECORD] = 1;
: 1257      1316 2
: 1258      1317 2  | If the target primary data record can not be completely deleted either
: 1259      1318 2  | because duplicates primary keys are allowed or it is the last record
: 1260      1319 2  | in the bucket, mark the record deleted, and squish out the data portion
: 1261      1320 2  | of the primary data record if it is squishable.
: 1262      1321 2
: 1263      1322 2  IF NOT .REC_ADDR[IRC$V_DELETED]
: 1264      1323 2  AND
: 1265      1324 3  (.IDX_DFN[IDX$V_DUPKEYS]
: 1266      1325 3  OR
: 1267      1326 3  .FLAGS[LAST_RECORD])
: 1268      1327 2  THEN
: 1269      1328 3  BEGIN
: 1270      1329 3  RM$SQUISH DATA();
: 1271      1330 3  REC_ADDR[IRC$V_DELETED] = 1;
: 1272      1331 3  RETURN;
: 1273      1332 3  END
: 1274      1333 3
: 1275      1334 3  | The primary data record can be completely deleted. It is either marked
: 1276      1335 3  | deleted (the only reason why RMS would be calling this routine would be
: 1277      1336 3  | to eliminate it entirely), or duplicates are not allowed and it is not
: 1278      1337 3  | the last primary data record in the bucket.
: 1279      1338 3
: 1280      1339 2  ELSE
: 1281      1340 3  BEGIN
: 1282      1341 3
: 1283      1342 3  LOCAL
: 1284      1343 3  UDR_ID;
: 1285      1344 3
: 1286      1345 3  | If the file is a prologue 1 or 2 file and the primary data record to
: 1287      1346 3  | be deleted is in its original bucket (ie - there is no RRV for it),
: 1288      1347 3  | then a two-byte RRV will have to be created for it at the end of the
: 1289      1348 3  | bucket inorder to reserve its ID and prevent it from being recycled.
: 1290      1349 3
: 1291      1350 3  AP = 3;
: 1292      1351 3
: 1293      1352 4  IF (.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
: 1294      1353 3  AND
: 1295      1354 4  (RM$RECORD_VBN() EQLA .BDB[BDB$L_VBN])
: 1296      1355 3  THEN
: 1297      1356 4  BEGIN
: 1298      1357 4  FLAGS[BUILD_RRV] = 1;
: 1299      1358 4  UDR_ID = .REC_ADDR[IRC$B_ID];
: 1300      1359 3  END;
: 1301      1360 3
: 1302      1361 3  | If primary key compression is enabled, and this primary data record
: 1303      1362 3  | is not the last record in the file, then the key of the following
: 1304      1363 3  | record, whose front compression is based on this record, will have
: 1305      1364 3  | to be re-expanded, after this target primary data record is
: 1306      1365 3  | completely removed. Set the local state bit accordingly and save the
: 1307      1366 3  | entire key portion (both control bytes and key) of the target primary
: 1308      1367 3  | data record in keybuffer 5 to be used in re-expanded the key of the
```



			2C BB 00000 RM\$DELETE UDR::			
			PUSHR #^M<R2,R3,R5>			1141
05	07	5E	SUBL2 #12, SP			1269
		A9	BBC #5, 7(IRAB), 1\$			1272
		66	BISB2 #32, (REC_ADDR)			1271
05	07	A9	BRB 6\$			1281
		66	BBC #6, 7(IRAB), 2\$			1284
			BICB2 #32, (REC_ADDR)			1283
			BRB 6\$			1291
			CLRB FLAGS			1292
			MOVL 24(BDB), BKT_ADDR			1293
			MOVZWL 4(BKT_ADDR), END_OF_BUCKET			1304
			ADDL2 BKT_ADDR, END_OF_BUCKET			1305
			CLRL R1			1311
			BSBW RMSREC OVHD			1313
50	04	AE	MOVL R0, REC OVHD			1315
6E	56		ADDL3 REC_OVHD, REC_ADDR, R0			1322
	50		ADDL3 REC_SIZE, R0, NEXT_REC_ADDR			1324
	52		CMPL NEXT_REC_ADDR, END_OF_BUCKET			1326
			BEQL 3\$			1328
03	00	BE	BBC #3, @NEXT_REC_ADDR, 4\$			1329
		53	BISB2 #2, FLAGS			1330
11	66		BBS #2, (REC_ADDR), 7\$			1332
09	04		BLBS 28(IDX_DFN), 5\$			1334
	53		BBC #1, FLAGS, 7\$			1336
			BSBW RM\$SQUISH DATA			1339
			BISB2 #4, (REC_ADDR)			1350
			BRW 12\$			1352
			MOVL #3, AP			1354
			CMPB 183(IFAB), #3			1357
			BGEQU 8\$			1358
			BSBW RMSRECORD VBN			1360
			CMPL R0, 28(BDB)			1370
			BNEQ 8\$			1372
22	08	53	BISB2 #1, FLAGS			1375
1E	1C	AE	MOVZBL 1(REC_ADDR), UDR_ID			1377
	1C	A7	01 A6 9A 00070			1378
		53	06 E1 00075	8\$:		1379
		53	01 E0 0007A			1380
		53	04 88 0007E			1382
		50	00B4 CA 3C 00081			1384
			60 B940 DF 00086			1386
			08 BE46 9F 0008A			1388
		7E	0C BE46 9A 0008E			1390
		6E	02 C0 00093			1392
			0000G 30 00096			1394
		5E	OC C0 00099			1396
		52	6E D1 0009C	9\$:		1398
			10 1E 0009F			1400
			56 DD 000A1			1402
		04	PUSHL REC_ADDR			1404
7E	52	08	AE C3 000A3			1406
			0000G 30 000AB			1408
		5E	OC C0 000AE			1410
			ADDL2 #12, SP			1412

50	56	6E	C3	000B1	10\$:	SUBL3	NEXT REC ADDR, REC_ADDR, R0	
	A5	50	A0	000B5		ADDW2	RO, 4(BKT_ADDR)	1396
12	53	02	E1	000B9		BBC	#2, FLAGS, 11\$	1403
51	56	04	AE	C1	000BD	ADDL3	REC OVHD, REC_ADDR, R1	1405
	50	00B4	CA	3C	000C2	MOVZWL	1807(IFABS) RO	
	50	60	B940	DE	000C7	MOVAL	@96(IRAB)[RO], RO	
			0000G	30	000CC	BSBW	RM\$EXPAND KEYD	
	13		53	E9	000CF	BLBC	FLAGS, 12\$	1411
	52	04	A5	3C	000D2	MOVZWL	4(BKT_ADDR), END_OF_BUCKET	1414
	52		55	C0	000D6	ADDL2	BKT_ADDR, END_OF_BUCKET	
	62		1C	90	000D9	MOVB	#28, (END_OF_BUCKET)	1416
01	A2	08	AE	90	000DC	MOVB	UDR_ID, 17END_OF_BUCKET)	1417
04	A5	02	A0	000E1		ADDW2	#2, 4(BKT_ADDR)	1418
	5E	0C	C0	000E5	12\$:	ADDL2	#12, SP	1423
		2C	BA	000E8		POPR	#^M<R2,R3,R5>	
			05	000EA		RSB		

; Routine Size: 235 bytes, Routine Base: RM\$RMS3 + 0274

```
1366 1424 1 %SBTTL 'RMSSQUISH DATA'  
1367 1425 1 ROUTINE RMSSQUISH_DATA : RL$SQUISH_DATA NOVALUE =  
1368 1426 1  
1369 1427 1 ++  
1370 1428 1  
1371 1429 1 FUNCTIONAL DESCRIPTION:  
1372 1430 1  
1373 1431 1 This routine's responsibility is the deletion of the data part of  
1374 1432 1 the current primary data record. This deletion can only take place if  
1375 1433 1 the file is a prologue 3 file.  
1376 1434 1  
1377 1435 1 CALLING SEQUENCE:  
1378 1436 1  
1379 1437 1 RMSSQUISH_DATA()  
1380 1438 1  
1381 1439 1 INPUT PARAMETERS:  
1382 1440 1 NONE  
1383 1441 1  
1384 1442 1 IMPLICIT INPUTS:  
1385 1443 1  
1386 1444 1 BKT_ADDR - address of the primary data bucket  
1387 1445 1  
1388 1446 1  
1389 1447 1 IDX_DFN - address of the primary key index descriptor  
1390 1448 1 IDX$V_KEY_COMP - if set, key compression is enabled  
1391 1449 1 IDX$B_KEYSZ - size of the key  
1392 1450 1 IDX$V_REC_COMP - if set, record compression is enabled  
1393 1451 1  
1394 1452 1 IFAB - address of the IFAB  
1395 1453 1 IFB$B_PLG_VER - prologue version of the file  
1396 1454 1  
1397 1455 1 REC_ADDR - address of the current primary data record  
1398 1456 1  
1399 1457 1 OUTPUT PARAMETERS:  
1400 1458 1 NONE  
1401 1459 1  
1402 1460 1 IMPLICIT OUTPUTS:  
1403 1461 1 NONE  
1404 1462 1  
1405 1463 1 ROUTINE VALUE:  
1406 1464 1 NONE  
1407 1465 1  
1408 1466 1 SIDE EFFECTS:  
1409 1467 1 The freespace in the bucket is updated to reflect the space reclaimed.  
1410 1468 1  
1411 1469 1 --  
1412 1470 2 BEGIN  
1413 1471 2  
1414 1472 2 EXTERNAL REGISTER  
1415 1473 2 R_BKT_ADDR STR,  
1416 1474 2 R_IDX_DFN STR,  
1417 1475 2 R_IFAB STR,  
1418 1476 2 R_REC_ADDR_STR;  
1419 1477 2  
1420 1478 2 GLOBAL REGISTER  
1421 1479 2 R_RAB,  
1422 1480 2 R_IRAB,
```

```
1423 1481 2 R_IMPURE,  
1424 1482 2 R_BDB;  
1425 1483 2  
1426 1484 2 LOCAL  
1427 1485 2 REC_SIZE,  
1428 1486 2 KEY_SIZE,  
1429 1487 2 REC_OVHD,  
1430 1488 2 SIZE;  
1431 1489 2  
1432 1490 2 | If this is not a prologue 3 file then nothing can be done; however, if  
1433 1491 2 this is a prologue 3 file then as the primary key is always kept separate  
1434 1492 2 from the data portion of a prologue 3 data record, the data portion  
1435 1493 2 of the current primary data record can always be squished out, and its  
1436 1494 2 space recovered.  
1437 1495 2  
1438 1496 2 IF .IFAB[IFB$B_PLG_VER] NEQ PLG$C_VER_3  
1439 1497 2 THEN  
1440 1498 2 RETURN;  
1441 1499 2  
1442 1500 2 Obtain the size of the record overhead and the size of the current  
1443 1501 2 primary data record. Note that the size of the key (and any key specific  
1444 1502 2 control bytes) is always included as part of the size of the current  
1445 1503 2 primary data record.  
1446 1504 2  
1447 1505 2 REC_OVHD = RMSREC_OVHD(0; REC_SIZE);  
1448 1506 2  
1449 1507 2 Compute the contribution of the primary key of the record to the size of  
1450 1508 2 the current primary data record. If primary key compression is enabled,  
1451 1509 2 then the key size will include the two bytes of key compression overhead.  
1452 1510 2  
1453 1511 2 IF .IDX_DFN[IDX$V_KEY_COMPR]  
1454 1512 2 THEN  
1455 1513 2 KEY_SIZE = .(REC_ADDR + .REC_OVHD)<0,8> + 2  
1456 1514 2 ELSE  
1457 1515 2 KEY_SIZE = .IDX_DFN[IDX$B_KEYSZ];  
1458 1516 2  
1459 1517 2 Compute the size of the data portion of the current primary data record.  
1460 1518 2 If the current primary data record consists of the primary key alone,  
1461 1519 2 return, as there is no data portion to squish out.  
1462 1520 2  
1463 1521 3 IF ((SIZE = .REC_SIZE - .KEY_SIZE) EQLU 0)  
1464 1522 2 THEN  
1465 1523 2 RETURN;  
1466 1524 2  
1467 1525 2 | Squish out the data portion of the current primary data record.  
1468 1526 2  
1469 1527 3 RMSMOVE ((.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])  
1470 1528 2 - (.REC_ADDR + .REC_OVHD + .REC_SIZE),  
1471 1529 2 .REC_ADDR + .REC_OVHD + .REC_SIZE  
1472 1530 2 .REC_ADDR + .REC_OVHD + .KEY_SIZE);  
1473 1531 2  
1474 1532 2 | Update the record size of the current primary data record to reflect  
1475 1533 2 the squishing out of the data portion of the record. NOTE that if the  
1476 1534 2 record is fixed length and both key and record compression are disabled,  
1477 1535 2 then there will be no record size field to update.  
1478 1536 2  
1479 1537 3 IF NOT (.IFAB[IFB$B_RFMOORG] EQLU FAB$C_FIX
```

```

1480 1538 3
1481 1539 3
1482 1540 3
1483 1541 3
1484 1542 2
1485 1543 2
1486 1544 2
1487 1545 2
1488 1546 2
1489 1547 2
1490 1548 2
1491 1549 2
1492 1550 2
1493 1551 2
1494 1552 1

      AND
      NOT .IDX_DFN[IDX$V_KEY_COMPR]
      AND
      NOT .IDX_DFN[IDX$V_REC_COMPR])
      THEN
        (.REC_ADDR + .REC_OVHD - 2)<0,16> = (.REC_ADDR + .REC_OVHD - 2)<0,16>
                                         - .SIZE;
      | Update the freespace pointer in the bucket to reflect the space that
      | has been recovered by the squishing out of the data portion of the
      | current primary data record.
      BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE] - .SIZE;
      END;

```

0B1C 8F BB 00000 RMSSQUISH DATA:							
					PUSHR	#^M<R2,R3,R4,R8,R9,R11>	1425
		03	00B7	CA 91 00004	CMPB	183(IFAB), #3	1496
				58 12 00009	BNEQ	5\$	1505
				51 D4 0000B	CLRL	R1	
				0000G 30 0000D	BSBW	RM\$REC_OVHD	
				50 D0 00010	MOVL	R0, REC_OVHD	
09	1C	52		06 E1 00013	BBC	#6, 28(IDX_DFN), 1\$	1511
		A7		6246 9A 00018	MOVZBL	(REC_OVHD)[REC_ADDR], KEY_SIZE	1513
		54		02 C0 0001C	ADDL2	#2, KEY_SIZE	
		54		04 11 0001F	BRB	2\$	
		54	20	A7 9A 00021 1\$:	MOVZBL	32(IDX_DFN), KEY_SIZE	1515
		51		54 C3 00025 2\$:	SUBL3	KEY_SIZE, REC_SIZE, SIZE	1521
		50	56	38 13 00029	BEQL	5\$	
				52 C1 0002B	ADDL3	REC_OVHD, REC_ADDR, R0	1530
				6440 9F 0002F	PUSHAB	(KEY_SIZE)[R0]	
			51	50 C0 00032	ADDL2	R0, R1	1529
				51 DD 00035	PUSHL	R1	
			50	04 A5 3C 00037	MOVZWL	4(BKT_ADDR), R0	1527
			50	55 C0 0003B	ADDL2	BKT_ADDR, R0	
	7E	50		51 C3 0003E	SUBL3	R1, R0, -(SP)	1528
				0000G 30 00042	BSBW	RM\$MOVE	
			5E	0C C0 00045	ADDL2	#12, SP	
		01	50	AA 91 00048	CMPB	80(IFAB), #1	1537
				0A 12 0004C	BNEQ	3\$	
05	1C	A7	06	E0 0004E	BBS	#6, 28(IDX_DFN), 3\$	1539
			1C	A7 95 00053	TSTB	28(IDX_DFN)	1541
				07 18 00056	BGEQ	4\$	
		04	9E	FE A246 9F 00058 3\$:	PUSHAB	-2(REC_OVHD)[REC_ADDR]	1544
		A5		53 A2 0005C	SUBW2	SIZE, -(SP)+	
				53 A2 0005F 4\$:	SUBW2	SIZE, 4(BKT_ADDR)	1550
			0B1C	8F BA 00063 5\$:	POPR	#^M<R2,R3,R4,R8,R9,R11>	
				05 00067	RSB		1552

: Routine Size: 104 bytes, Routine Base: RMSRMS3 + 035F

```
1496 1553 1 %SBTTL 'RMSSQUISH_SIDR'  
1497 1554 1 GLOBAL ROUTINE RMSSQUISH_SIDR (SCAN, BEGIN_OF_SIDR) : RL$RABREG_567 =  
1498 1555 1  
1499 1556 1 ++  
1500 1557 1  
1501 1558 1 FUNCTIONAL DESCRIPTION:  
1502 1559 1  
1503 1560 1 This routine's responsibility is to delete the SIDR array element  
1504 1561 1 pointing to the current primary data record for this key of reference.  
1505 1562 1 Deletion of the SIDR array element goes according to one of the  
1506 1563 1 following rules:  
1507 1564 1  
1508 1565 1 1. Removal of the entire SIDR if duplicates are not allowed. NOTE that  
1509 1566 1 if the input parameter SCAN is 1 and the file is a prologue 3 file  
1510 1567 1 then for the purpose of this SIDR deletion it is assumed that this  
1511 1568 1 key of reference does allow duplicates (See rules 2 through 5).  
1512 1569 1  
1513 1570 1 2. Marking the SIDR array element as deleted and not recovering any  
1514 1571 1 space if duplicates are allowed for this key of reference and the  
1515 1572 1 file is a prologue 1 or 2 file.  
1516 1573 1  
1517 1574 1 3. Marking the SIDR array element as deleted and not recovering any  
1518 1575 1 space if duplicates are allowed for this key of reference, the file  
1519 1576 1 is a prologue 3 file, and the element is the first element in the  
1520 1577 1 SIDR array.  
1521 1578 1  
1522 1579 1 4. Marking the SIDR element deleted and squishing out the space  
1523 1580 1 occupied by the RRV pointer if duplicates are allowed for this key  
1524 1581 1 of reference, the file is a prologue 3 file, and the element is not  
1525 1582 1 the first element in the SIDR array.  
1526 1583 1  
1527 1584 1 5. Removal of the entire SIDR array if duplicates are allowed, this is  
1528 1585 1 the first SIDR with this key value, the SIDR is not the physically  
1529 1586 1 last SIDR in the bucket, and every single element within the SIDR  
1530 1587 1 array has been deleted.  
1531 1588 1  
1532 1589 1 6. If the state bit IRBSV_RU_DELETE is set, then the SIDR array element  
1533 1590 1 is just marked RU_DELETE and no space is reclaimed.  
1534 1591 1  
1535 1592 1 7. If the state bit IRBSV_RU_UNDEL is set, then the SIDR array element  
1536 1593 1 is un-deleted by clearing the RU_DELETE bit within the element's  
1537 1594 1 control byte.  
1538 1595 1  
1539 1596 1  
1540 1597 1  
1541 1598 1  
1542 1599 1  
1543 1600 1  
1544 1601 1  
1545 1602 1  
1546 1603 1  
1547 1604 1  
1548 1605 1  
1549 1606 1  
1550 1607 1  
1551 1608 1  
1552 1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
1603 1  
1604 1  
1605 1  
1606 1  
1607 1  
1608 1  
1609 1  
1553 1  
1554 1  
1555 1  
1556 1  
1557 1  
1558 1  
1559 1  
1560 1  
1561 1  
1562 1  
1563 1  
1564 1  
1565 1  
1566 1  
1567 1  
1568 1  
1569 1  
1570 1  
1571 1  
1572 1  
1573 1  
1574 1  
1575 1  
1576 1  
1577 1  
1578 1  
1579 1  
1580 1  
1581 1  
1582 1  
1583 1  
1584 1  
1585 1  
1586 1  
1587 1  
1588 1  
1589 1  
1590 1  
1591 1  
1592 1  
1593 1  
1594 1  
1595 1  
1596 1  
1597 1  
1598 1  
1599 1  
1600 1  
1601 1  
1602 1  
16
```

1553	1610	1	IDX_DFN	- address of the index descriptor
1554	1611	1	IDX\$V_DUPKEYS	- if set, duplicate keys are allowed
1555	1612	1	IDX\$V_KEY_COMPR	- if set, SIDR key compression is enabled
1556	1613	1	IFAB	
1557	1614	1	IFBSW_KBUFSZ	- address of IFAB
1558	1615	1	IFBSB_PLG_VER	- size of one of the contiguous keybuffers
1559	1616	1	IRAB	- prologue version of file
1560	1617	1	IRBSL_KEYBUF	
1561	1618	1	IRBSV_RU_DELETE	- address of IRAB
1562	1619	1	IRBSV_RU_UNDEL	- address of the contiguous keybuffers
1563	1620	1	REC_ADDR	- if set, mark RU_DELETE and do not reclaim
1564	1621	1		- if set, un-delete the RRV
1565	1622	1		
1566	1623	1		- address of the SIDR array element
1567	1624	1	OUTPUT PARAMETERS:	
1568	1625	1	NONE	
1569	1626	1	IMPLICIT OUTPUTS:	
1570	1627	1	REC_ADDR	- address of next SIDR if the entire SIDR was deleted
1571	1628	1		otherwise unchanged.
1572	1629	1	ROUTINE VALUE:	
1573	1630	1	1	- some space was recovered.
1574	1631	1	0	- no space was recovered.
1575	1632	1	SIDE EFFECTS:	
1576	1633	1	1640	Keybuffer 5 will have been trashed, if any key re-expansion occurred.
1577	1634	1	1641	The freespace in the bucket is updated to reflect the space reclaimed.
1578	1635	1	1642	If the SIDR is completely deleted, SIDR key compression is enabled, and
1579	1636	1	1643	a SIDR follows the completely deleted SIDR, then the key of this
1580	1637	1	1644	following SIDR will have been re-expanded.
1581	1638	1	1645	--
1582	1639	1	1646	BEGIN
1583	1640	1	1647	
1584	1641	1	1648	EXTERNAL REGISTER
1585	1642	1	1649	R_BKT_ADDR_STR,
1586	1643	1	1650	COMMON_RAB_STR,
1587	1644	1	1651	R_IDX_DFN_STR,
1588	1645	1	1652	R_REC_ADDR_STR:
1589	1646	1	1653	
1590	1647	1	1654	LABEL
1591	1648	2	1655	DUPS;
1592	1649	2	1656	LOCAL
1593	1650	2	1657	DELETE_START,
1594	1651	2	1658	DELETE_END,
1595	1652	2	1659	FLAGS : BLOCK[1],
1596	1653	2	1660	LENGTH,
1597	1654	2	1661	NEXT_REC_ADDR,
1598	1655	2	1662	RECORD_OVHD,
1599	1656	2	1663	SAVE_REC_ADDR : REF_BBLOCK;
1600	1657	2	1664	
1601	1658	2	1665	
1602	1659	2	1666	
1603	1660	2		
1604	1661	2		
1605	1662	2		
1606	1663	2		
1607	1664	2		
1608	1665	2		
1609	1666	2		

```
1610      1667  2
1611      1668  2
1612      1669  2
1613      1670  2
1614      1671  2
1615      1672  2
1616      1673  2
1617      1674  2
1618      1675  2
1619      1676  2
1620      1677  2
1621      1678  2
1622      1679  2
1623      1680  2
1624      1681  2
1625      1682  3
1626      1683  3
1627      1684  3
1628      1685  3
1629      1686  3
1630      1687  3
1631      1688  3
1632      1689  3
1633      1690  2
1634      1691  2
1635      1692  2
1636      1693  3
1637      1694  3
1638      1695  3
1639      1696  2
1640      1697  2
1641      1698  2
1642      1699  2
1643      1700  2
1644      1701  2
1645      1702  2
1646      1703  2
1647      1704  2
1648      1705  2
1649      1706  2
1650      1707  3
1651      1708  3
1652      1709  3
1653      1710  3
1654      1711  3
1655      1712  3
1656      1713  3
1657      1714  3
1658      1715  2
1659      1716  2
1660      1717  2
1661      1718  2
1662      1719  2
1663      1720  2
1664      1721  2
1665      1722  2
1666      1723  3

      MAP
        BEGIN_OF_SIDR : REF BBLOCK;
      MACRO
        DELETE_SIDR    = 0,0,1,0 %;
        SQUISH_SIDR   = 0,1,1,0 %;
        RE_EXPAND_KEY = 0,2,1,0 %;
      | If it is indicated that the SIDR array element should just be marked
      | RU_DELETE and that no space should be reclaimed, then do so by setting
      | the RU_DELETE bit within the element's control byte.
      IF .IRAB[IRB$V_RU_DELETE]
      THEN
        BEGIN
          REC_ADDR[IRC$V_RU_DELETE] = 1;
          RETURN 0;
        END
      | If it is indicated that the SIDR array element should be un-deleted,
      | then do so by clearing the RU_DELETE bit in the element's control byte.
      ELSE
        IF .IRAB[IRB$V_RU_UNDEL]
        THEN
          BEGIN
            REC_ADDR[IRC$V_RU_DELETE] = 0;
            RETURN 0;
          END;
      | Save the address of the current SIDR element, and zero out the local
      | flag field.
      ! FLAGS = 0;
      SAVE_REC_ADDR = .REC_ADDR;
      | Determine the address of the first byte past the end of the current
      | SIDR.
      BEGIN
        LOCAL
          REC_SIZE;
        REC_ADDR = .BEGIN_OF_SIDR;
        RECORD_OVHD = RMS$REC_OVHD(-1; REC_SIZE);
        NEXT_REC_ADDR = .REC_ADDR + .RECORD_OVHD + .REC_SIZE;
      END;
      | If this secondary key of reference does not allow duplicate key values
      | and either the file's prologue version is 1 or 2; or, the input parameter
      | SCAN is 0, then the entire SIDR maybe deleted.
      IF NOT .IDX_DFN[IDX$V_DUPKEYS]
        AND
        (NOT .SCAN
```

```
1667      1724 3          OR
1668      1725 3          .IFAB[IFBSB_PLG_VER] LSSU PLGSC_VER_3)
1669      1726 2          THEN
1670      1727 2          FLAGS[DELETE_SIDR] = 1
1671      1728 2
1672      1729 2          If this key of reference does allow duplicate SIDR keys or duplicates are
1673      1730 2          not allowed but the file's prologue version is 3 and a scan of then entire
1674      1731 2          SIDR array has been requested (SCAN is set to 1), then mark the current
1675      1732 2          element as deleted and under certain circumstances, reclaim the space
1676      1733 2          occupied by the SIDR array element's RRV pointer. Under very restricted
1677      1734 2          circumstances it will also be possible to reclaim the space occupied by
1678      1735 2          the entire SIDR.
1679      1736 2
1680      1737 2          ELSE
1681      1738 2          DUPs:
1682      1739 3          BEGIN
1683      1740 3          SAVE_REC_ADDR[IRC$V_DELETED] = 1;
1684      1741 3
1685      1742 3          If the file is a prologue 2 file then marking the element deleted is
1686      1743 3          all that can be done.
1687      1744 3
1688      1745 4          IF (.IFAB[IFBSB_PLG_VER] LSSU PLGSC_VER_3)
1689      1746 3          THEN
1690      1747 4          BEGIN
1691      1748 4          REC ADDR = .SAVE_REC_ADDR;
1692      1749 4          RETURN 0;
1693      1750 4          END
1694      1751 4
1695      1752 4          The file is a prologue 3 file. If every single array element in this
1696      1753 4          SIDR array is deleted, if the SIDR is not physically the last SIDR in
1697      1754 4          the bucket (this restriction applies to duplicates keys allowed only)
1698      1755 4          and if this SIDR is the first such SIDR with this key value in the
1699      1756 4          file then it will be possible to delete the entire SIDR; otherwise,
1700      1757 4          the space occupied by the element's RRV pointer is reclaimed unless
1701      1758 4          it is the first element in the array in which case nothing more can
1702      1759 4          be done.
1703      1760 4
1704      1761 3          ELSE
1705      1762 4          BEGIN
1706      1763 4
1707      1764 4          LABEL
1708      1765 4          ENTIRE_SIDR;
1709      1766 4
1710      1767 4
1711      1768 4          LOCAL
1712      1769 4          FIRST_SIDR : REF BBLOCK;
1713      1770 4          Obtain the address of the first array element in the SIDR array.
1714      1771 4
1715      1772 4          FIRST_SIDR = RM$SIDR_FIRST(0);
1716      1773 4
1717      1774 4          If the first element in the array (which maybe the element being
1718      1775 4          deleted) is marked deleted, and this SIDR is the first such
1719      1776 4          record in the file with this key value, then it still maybe
1720      1777 4          possible to delete the entire SIDR.
1721      1778 4
1722      1779 4          IF .FIRST_SIDR[IRC$V_DELETED]
1723      1780 4          AND
```

```
1724      1781 4      .FIRST_SIDR[IRC$V_FIRST_KEY]
1725      1782 4
1726      1783 5 ENTIRE_SIDR: THEN BEGIN
1727      1784 5
1728      1785 5      LOCAL
1729      1786 5      SCAN_START;
1730      1787 5
1731      1788 5      | If the current SIDR is physically the last SIDR in the bucket
1732      1789 5      | and duplicates keys are allowed then it will not be possible
1733      1790 5      | to reclaim the space occupied by the entire SIDR even if all
1734      1791 5      | its elements are deleted.
1735      1792 5
1736      1793 6      IF .NEXT_REC_ADDR GEQA (.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
1737      1794 5      AND
1738      1795 5      .IDX_DFN[IDX$V_DUPKEYS]
1739      1796 5 THEN LEAVE ENTIRE_SIDR;
1740      1797 5
1741      1798 5
1742      1799 5      | Scan the SIDR array starting with the second element up to
1743      1800 5      | but not including the target element making sure that all
1744      1801 5      | these elements have been deleted. If a live element is found
1745      1802 5      | then the space occupied by the entire SIDR can not be
1746      1803 5      | reclaimed.
1747      1804 5
1748      1805 5      SCAN_START = .FIRST_SIDR + .FIRST_SIDR[IRC$V_PTRSZ]
1749      1806 5      + IRC$C_DATPTRBS3
1750      1807 5      + 1;
1751      1808 5
1752      1809 6      IF (.SCAN_START LSSA .SAVE_REC_ADDR)
1753      1810 5 THEN IF NOT CH$FAIL (CH$FIND NOT CH
1754      1811 5      (.SAVE_REC_ADDR - .SCAN_START,
1755      1812 5      .SCAN_START,
1756      1813 5      %CHAR(IRC$M_DELETED)
1757      1814 5      OR
1758      1815 5      %CHAR(IRC$M_NOPTRSZ)))
1759      1816 5
1760      1817 5 THEN LEAVE ENTIRE_SIDR;
1761      1818 5
1762      1819 5
1763      1820 5      | Scan the SIDR array starting with the first element past the
1764      1821 5      | target element and ending with the last element in the SIDR
1765      1822 5      | making sure that all these elements have been deleted. If a
1766      1823 5      | live element is found then the space occupied by the entire
1767      1824 5      | SIDR can not be reclaimed.
1768      1825 5
1769      1826 5      SCAN_START = .SAVE_REC_ADDR + .SAVE_REC_ADDR[IRC$V_PTRSZ]
1770      1827 5      + IRC$C_DATPTRBS3
1771      1828 5      + 1;
1772      1829 5
1773      1830 6      IF (.SCAN_START LSSA .NEXT_REC_ADDR)
1774      1831 5 THEN IF NOT CH$FAIL (CH$FIND NOT CH
1775      1832 5      (.NEXT_REC_ADDR - .SCAN_START,
1776      1833 5      .SCAN_START,
1777      1834 5      %CHAR(IRC$M_DELETED)
1778      1835 5      OR
1779      1836 5      %CHAR(IRC$M_NOPTRSZ)))
1780      1837 5
```

```
1781      1838 5      THEN      LEAVE ENTIRE_SIDR;
1782      1839 5
1783      1840 5
1784      1841 5      | Every single element in the current SIDR has been found to be
1785      1842 5      | deleted, so the space occupied by the entire SIDR maybe
1786      1843 5      | reclaimed.
1787      1844 5
1788      1845 5      FLAGS[DELETE_SIDR] = 1;
1789      1846 5      LEAVE DUPS;
1790      1847 4      END;
1791      1848 4
1792      1849 4      | If it is not possible to delete the entire SIDR then set up to
1793      1850 4      | reclaim the space occupied by the element's RRV pointer unless the
1794      1851 4      | element is the first element in the array in which case nothing
1795      1852 4      | more can be done.
1796      1853 4
1797      1854 4      REC_ADDR = .SAVE_REC_ADDR;
1798      1855 4
1799      1856 5      IF (.REC_ADDR EQA .FIRST_SIDR)
1800      1857 4      THEN
1801      1858 4      RETURN 0
1802      1859 4      ELSE
1803      1860 4      FLAGS[SQUISH_SIDR] = 1;
1804      1861 3      END;
1805      1862 2
1806      1863 2
1807      1864 2      | If the space occupies by the entire SIDR is to be reclaimed, set up to
1808      1865 2      | recover it.
1809      1866 2
1810      1867 2      IF .FLAGS[DELETE_SIDR]
1811      1868 2      THEN
1812      1869 3      BEGIN
1813      1870 3      DELETE_START = .BEGIN_OF_SIDR;
1814      1871 3      DELETE_END = .NEXT_REC_ADDR;
1815      1872 3
1816      1873 3      | If key compression is enabled, and this SIDR is not the last SIDR
1817      1874 3      | in the bucket, save the key of the current SIDR in keybuffer 5,
1818      1875 3      | so that it maybe used in expanding the key of the following
1819      1876 3      | record.
1820      1877 3
1821      1878 3      IF .IDX_DFN[IDX$V_KEY_COMPR]
1822      1879 3      THEN
1823      1880 4      BEGIN
1824      1881 4
1825      1882 4      GLOBAL REGISTER
1826      1883 4      R_BDB;
1827      1884 4
1828      1885 4      FLAGS[RE_EXPAND_KEY] = 1;
1829      1886 4
1830      1887 4      RMSMOVE (.REC_ADDR + .RECORD_OVHD)<0,8> + 2,
1831      1888 4      .REC_ADDR + .RECORD_OVHD,
1832      1889 4      KEYBUF_ADDR($));
1833      1890 3      END;
1834      1891 3      END
1835      1892 3
1836      1893 3      | If the space occupies by the RRV pointer is to be reclaimed, set up to
1837      1894 3      | recover it.
```

```
1838 1895 3      !
1839 1896 2      ELSE
1840 1897 3      BEGIN
1841 1898 3
1842 1899 3      DELETE_START = .REC_ADDR + 1;
1843 1900 3      DELETE_END   = .DELETE_START + .REC_ADDR[IRC$V_PTRSZ]
1844 1901 3      + IRC$C_DATPTRBS3;
1845 1902 3
1846 1903 3      REC_ADDR[IRC$V_NOPTRSZ] = 1;
1847 1904 3      REC_ADDR[IRC$V_PTRSZ]   = 0;
1848 1905 3
1849 1906 3      ! Update the SIDR size field. As it is currently written, this
1850 1907 3      updating assumes that the size field is the first two bytes
1851 1908 3      (and the only two bytes) of the record overhead field.
1852 1909 3
1853 1910 3      (.BEGIN_OF_SIDR)<0,16> = .(.BEGIN_OF_SIDR)<0,16>
1854 1911 3      - (.DELETE_END - .DELETE_START);
1855 1912 2      END;
1856 1913 2
1857 1914 2      ! Recover the space that can be recovered, and update the freespace offset
1858 1915 2      in the SIDR bucket. If the SIDR is being completely deleted, and it is the
1859 1916 2      last SIDR in the bucket then there will be nothing to move and only the
1860 1917 2      bucket's freespace offset need be updated.
1861 1918 2
1862 1919 2      LENGTH = .BKT_ADDR[BKT$W_FREESPACE] - (.DELETE_END - .BKT_ADDR);
1863 1920 2
1864 1921 2      IF .LENGTH GTRU 0
1865 1922 2      THEN
1866 1923 3      BEGIN
1867 1924 3
1868 1925 3      GLOBAL REGISTER
1869 1926 3      R_BDB;
1870 1927 3
1871 1928 3      RM$MOVE (.LENGTH, .DELETE_END, .DELETE_START);
1872 1929 2      END;
1873 1930 2
1874 1931 2      BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
1875 1932 2      - (.DELETE_END - .DELETE_START);
1876 1933 2
1877 1934 2      ! If key compression is enabled, the space occupied by the current SIDR was
1878 1935 2      completely reclaimed, and a SIDR follows whose key needs to be
1879 1936 2      re-expanded, do so at this point.
1880 1937 2
1881 1938 2      IF .FLAGS[RE_EXPAND_KEY]
1882 1939 2      THEN
1883 1940 2      RM$EXPAND_KEYD (KEYBUF_ADDR(5), .REC_ADDR + .RECORD_OVHD);
1884 1941 2
1885 1942 2      ! Return indicating that some space has been recovered.
1886 1943 2
1887 1944 2      RETURN 1;
1888 1945 1      END;
```

\*\*\*

05	07	5E	0C	C2 00002	PUSHR	#^M<R2,R3,R4>	: 1554
		A9	05	E1 00005	SUBL2	#12, SP	1680
		66	20	88 0000A	BBC	#5, 7(IRAB), 1\$	1683
			43	11 0000D	BISB2	#32, (REC_ADDR)	1684
05	07	A9	06	E1 0000F	1\$:	BRB	4\$
		66	20	8A 00014	BBC	#6, 7(IRAB), 2\$	
			39	11 00017	BICB2	#32, (REC_ADDR)	
			04	AE D4 00019	2\$:	BRB	4\$
			54	56 D0 0001C	CLRL	FLAGS	
			56	20 AE 0001F	MOVL	REC_ADDR, SAVE_REC_ADDR	
			51	01 CE 00023	MOVL	BEGIN_OF_SIDR, REC_ADDR	
				0000G 30 00026	MNEGL	#1 RT	
50	08	AE	50	D0 00029	BSBW	RM\$REC_OVHD	
53			08	AE C1 0002D	MOVL	RO, RECORD_OVHD	
			50	51 C1 00032	ADDL3	RECORD_OVHD, REC_ADDR, RO	
			0B	1C A7 E8 00036	ADDL3	REC_SIZE, R0, NEXT_REC_ADDR	
			7E	1C AE E9 0003A	BLBS	28(IDX_DFN), 3\$	
			03	00B7 CA 91 0003E	BLBC	SCAN, T0\$	
				77 1F 00043	CMPB	183(IFAB), #3	
			64	04 88 00045	BLSSU	10\$	
		03	00B7	CA 91 00048	BISB2	#4, (SAVE_REC_ADDR)	
				06 1E 0004D	CMPB	183(IFAB), #3	
			56	54 D0 0004F	BGEQU	5\$	
				0100 31 00052	MOVL	SAVE_REC_ADDR, REC_ADDR	
				7E D4 00055	BRW	17\$	
				0000G 30 00057	CLRL	-(SP)	
			5E	04 C0 0005A	BSBW	RM\$SIDR_FIRST	
5D	00	6E	50	D0 0005D	ADDL2	#4, SP	
		BE	00	BE 95 00060	MOVL	RO, FIRST_SIDR	
			50	58 18 00065	BBC	#2, @FIRST_SIDR, 11\$	
			50	04 A5 3C 0006A	TSTB	@FIRST_SIDR	
			50	55 C0 0006E	BGEQ	11\$	
			50	53 D1 00071	MOVZWL	4(BKT_ADDR), RO	
			50	04 1F 00074	ADDL2	BKT_ADDR, R0	
50	00	48	1C	A7 E8 00076	CMPL	NEXT_REC_ADDR, RO	
		BE	02	00 EF 0007A	BLSSU	6\$	
			51	6E D0 00080	BLBS	28(IDX_DFN), 11\$	
			52	05 A041 9E 00083	EXTZV	#0, #2, @FIRST_SIDR, RO	
			54	52 D1 00088	MOVAB	FIRST_SIDR, R1	
				10 1E 0008B	CMPL	5(RO)[R1], SCAN_START	
			50	54 52 C3 0008D	BGEQU	SCAN_START, SAVE_REC_ADDR	
		62	50	14 3B 00091	SUBL3	1809	
				02 12 00095	SKPC	SCAN_START, SAVE_REC_ADDR, RO	
			51	51 D4 00097	BNEQ	#20, RO, (SCAN_START)	
				25 12 0009B	CLRL	1812	
50	64	02	00	EF 0009D	7\$:	R1	
			52	05 A044 9E 000A2	TSTL	R1	
			53	52 D1 000A7	BNEQ	11\$	
				10 1E 000AA	EXTZV	1815	
			50	52 C3 000AC	MOVAB	#0, #2, (SAVE_REC_ADDR), RO	
		62	50	14 3B 000B0	CMPL	1826	
				02 12 000B4	BGEQU	5(RO)[SAVE_REC_ADDR], SCAN_START	
			51	51 D4 000B6	SUBL3	1828	
				25 12 000B8	SKPC	SCAN_START, NEXT_REC_ADDR, RO	
			51	51 D5 000B8	BNEQ	#20, RO, (SCAN_START)	
				9\$:	CLRL	1830	
					TSTL	R1	
						1833	
						1836	
						1837	

RM3DELETE  
V04-000

RM\$SQUISH\_SIDR

D 14  
16-Sep-1984 01:42:30 VAX-11 Bliss-32 V4.0-742  
14-Sep-1984 13:01:19 [RMS.SRC]RM3DELETE.B32;1

Page 45  
(8)

RM  
VO

04	AE		06	12	000BA		BNEQ	11\$					1845
			01	88	000BC	10\$:	BISB2	#1, FLAGS					1846
	56		0C	11	000CO		BRB	12\$					1854
	6E		54	D0	000C2	11\$:	MOVL	SAVE REC_ADDR, REC_ADDR					1856
			56	D1	000C5		CMPL	REC_ADDR, FIRST_SIDR					
			88	13	000C8		BEQL	4\$					
04	AE		02	88	000CA		BISB2	#2, FLAGS					1860
	2D	04	AE	E9	000CE	12\$:	BLBC	FLAGS, 13\$					1867
	51	20	AE	D0	000D2		MOVL	BEGIN_OF_SIDR, DELETE_START					1870
	52		53	D0	000D6		MOVL	NEXT_REC_ADDR, DELETE_END					1871
3D	1C	A7	06	E1	000D9		BBC	#6, 28(IDX_DFN), 14\$					1878
	04	AE	04	88	000DE		BISB2	#4, FLAGS					1885
		50	00B4	CA	3C	000E2	MOVZWL	180(IFAB), R0					1889
			60	B940	DF	000E7	PUSHAL	@96(IRAB)[R0]					
			0C	BE46	9F	000EB	PUSHAB	@RECORD_OVHD[REC_ADDR]					1888
		7E	10	BE46	9A	000EF	MOVZBL	@RECORD_OVHD[REC_ADDR], -(SP)					1887
		6E		02	C0	000F4	ADDL2	#2, (SP)					
				0000G	30	000F7	BSBW	RM\$MOVE					
		5E		0C	C0	000FA	ADDL2	#12, SP					
				0C	11	000FD	BRB	14\$					
50			51	01	A6	9E	000FF	13\$:	MOVAB	1(R6), DELETE_START			1867
	66		02		00	EF	00103		EXTZV	#0, #2, (REC_ADDR), R0			1899
		52	04	A041	9E	00108	MOVAB	4(R0)[DELETE_START], DELETE_END					1900
		66		10	88	0010D	BISB2	#16, (REC_ADDR)					1901
	50	66		03	8A	00110	BICB2	#3, (REC_ADDR)					1903
		51	51	52	C3	00113	SUBL3	DELETE_END, DELETE_START, R0					1904
	50	20	BE	50	A0	00117	ADDW2	RO, @BEGIN_OF_SIDR					1911
	50		55	52	C3	0011B	14\$:	SUBL3	DELETE_END, BKT_ADDR, R0				1919
			53	04	A5	3C	0011F	SUBL3	4(BKT_ADDR), R3				
			50		53	C0	00123	MOVZWL	R3_LENGTH				
					0A	13	00126	ADDL2	15\$				1921
					51	DD	00128	BEQL	DELETE_START				1928
					05	BB	0012A	PUSHL	#^M<R0,R2>				
					0000G	30	0012C	PUSHR	RM\$MOVE				
					0C	C0	0012F	BSBW	#12, SP				
12		5E	51	52	C2	00132	15\$:	ADDL2	DELETE_END, R1				1932
51	04	04	A5	51	A0	00135	SUBL2	R1, 4(BKT_ADDR)					
			AE	02	E1	00139	ADDW2	#2, FLAGS, 16\$					1938
		56	08	AE	C1	0013E	BBC	RECORD_OVHD, REC_ADDR, R1					1940
		50	00B4	CA	3C	00143	ADDL3	180(IFAB), R0					
		50	60	B940	DE	00148	MOVZWL	@96(IRAB)[R0], R0					
				0000G	30	0014D	MOVAL	RM\$EXPAND_KEYD					
			50		01	D0	00150	16\$:	BSBW	#1, R0			1944
					02	11	00153	MOVL	18\$				
					50	D4	00155	17\$:	BRB	RO			1945
		5E			0C	C0	00157	18\$:	CLRL	#12, SP			
					1C	BA	0015A	ADDL2	#^M<R2,R3,R4>				
						05	0015C	RSB	POPR				

; Routine Size: 349 bytes, Routine Base: RM\$RMS3 + 03C7

1889 1946 1  
1890 1947 1 END  
1891 1948 1  
1892 1949 0 ELUDOM

RM3DELETE  
V04-000

RM\$SQUISH\_SIDR

E 14

16-Sep-1984 01:42:30  
14-Sep-1984 13:01:19

VAX-11 Bliss-32 V4.0-742  
[RMS.SRC]RM3DELETE.B32;1

Page 46  
(8)

RM  
VO

PSECT SUMMARY

Name	Bytes	Attributes
RM\$RMS3	1316	NOVEC,NOWRT, RD , EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	-----	Symbols	-----	Pages	Processing
	Total	Loaded	Percent	Mapped	Time
\$_255\$DUA28:[RMS.OBJ]RMS.L32;1	3109	92	2	154	00:00.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:RM3DELETE/OBJ=OBJ\$:RM3DELETE MSRC\$:RM3DELETE/UPDATE=(ENH\$:RM3DELETE)

1893 1950 0  
Size: 1316 code + 0 data bytes  
Run Time: 00:33.6  
Elapsed Time: 01:00.1  
Lines/CPU Min: 3483  
Lexemes/CPU-Min: 15181  
Memory Used: 163 pages  
Compilation Complete

0324 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

